NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ASSESSING THE POTENTIAL VALUE OF SEMANTIC WEB TECHNOLOGIES IN SUPPORT OF MILITARY OPERATIONS

by

Samuel G. Chance
Marty G. Hagenston

September 2003

Thesis Advisor:	Alex Bordetsky
Second Reader:	Douglas P. Horner

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE *Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
September 2003

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE: Assessing the Potential Value of Semantic Web Technologies in Support of Military Operations

5. FUNDING NUMBERS

6. AUTHOR(S) Samuel G. Chance and Marty G. Hagenston

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Recent military operations have redefined the way modern warfare is waged. In a deliberate effort to achieve and retain information dominance and decision superiority, many innovative technologies have emerged to assist the human war fighter. Unquestionably, these technologies have generated resounding successes on the battlefield, the likes of which have never been seen. With all the success, however, there are still areas for improvement as the potential exists for further reducing already short sensor-to-shooter times.

The current World Wide Web (WWW) is largely a human-centric information space where humans exchange and interpret data ([2] Berners-Lee, 1, 1999). The Semantic Web (SWEB) is not a separate Web, but an extension of the current one in which content is given well-defined meaning, better enabling computers and people to work in cooperation (Berners-Lee et al). The result is the availability of the various backgrounds, experiences, and abilities of the contributing communities through the self-describing content populating the SWEB ([2] Berners-Lee, 1999). This thesis assesses current SWEB technologies that promise to make disparate data sources machine interpretable for use in the construction of actionable knowledge with the intent of further reducing sensor-to-shooter times.

The adoption of the SWEB will quietly be realized and soon machines will prove to be of greater value to war fighting. When machines are able to interpret and process content before human interaction and analysis begins, their value will be further realized. This off-loading, or delegation, will produce faster sensor-to-shooter times and assist in achieving the speed required to achieve victory on any battlefield.

14. SUBJECT TERMS Semantic Web, XML, OWL, DAML, RDF, Knowledge Base, Database, Jini, Java, Agents, Ontologies, CoABS, Data Sources, Knowledge Generation, Jess

15. NUMBER OF PAGES
289

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

THIS PAGE INTENTIONALLY LEFT BLANK

ASSESSING THE POTENTIAL VALUE OF SEMANTIC WEB TECHNOLOGIES IN SUPPORT OF MILITARY OPERATIONS

Samuel G. Chance
Lieutenant, United States Navy
B.S., Florida A&M University, 1995

Marty G. Hagenston
Major, United States Army
B.A., Washington State University, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 2003

Authors:	Samuel G. Chance
	Marty G. Hagenston
Approved by:	Alexander Bordetsky Thesis Advisor
	Douglas P. Horner Second Reader
	Dan C. Boger Chairman, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Recent military operations have redefined the way modern warfare is waged. In a deliberate effort to achieve and retain information dominance and decision superiority, many innovative technologies have emerged to assist the human war fighter. Unquestionably, these technologies have generated resounding successes on the battlefield, the likes of which have never been seen. With all the success, however, there are still areas for improvement as the potential exists for further reducing already short sensor-to-shooter times.

The current World Wide Web (WWW) is largely a human-centric information space where humans exchange and interpret data ([2] Berners-Lee, 1, 1999). The Semantic Web (SWEB) is not a separate Web, but an extension of the current one in which content is given well-defined meaning, better enabling computers and people to work in cooperation (Berners-Lee et al). The result is the availability of the various backgrounds, experiences, and abilities of the contributing communities through the self-describing content populating the SWEB ([2] Berners-Lee, 1999). This thesis assesses current SWEB technologies that promise to make disparate data sources machine interpretable for use in the construction of actionable knowledge with the intent of further reducing sensor to shooter times.

The adoption of the SWEB will quietly be realized and soon machines will prove to be of greater value to war fighting. When machines are able to interpret and process content before human interaction and analysis begins, their value will be further realized. This off-loading, or delegation, will produce faster sensor-to-shooter times and assist in achieving the speed required to achieve victory on any battlefield.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Let us run with perseverance the race marked out for us. (Hebrews 12:1b). I have completed this race and have persevered. Moreover, my family has endured much for me, and our nation. I thank my wife and all our children for their selflessness, sacrifice and love. I thank the Lord, above all, for allowing me this opportunity. I also thank the faculty for their flexibility and support. I sincerely appreciate Mr. Doug Horner's generosity, insight and support. Finally, I thank my thesis partner for setting such a good example of diligence, initiative and intelligence. His steadfast ability to acquire and assimilate information is most inspiring. It was truly an honor to be associated and partnered with him.

- Sam

A heart-felt thanks to my wife and children for the support and love they give me; to my research partner for his superior competence and passion; and to my advisors for their wisdom.

-Marty

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Recent military operations have redefined the way modern warfare is waged. In a deliberate effort to achieve and retain information dominance and decision superiority, many innovative technologies have emeasdfsdafasdfdasfrged to assist the human war fighter. Unquestionably, these technologies have generated resounding successes on the battlefield, the likes of which have never been seen. With all the success, however, there are still areas for improvement as the potential exists for further reducing already short sensor-to-shooter times.

This thesis assesses current Semantic Web (SWEB) technologies that promise to make disparate data sources machine interpretable for use in the construction of actionable knowledge with intent of further reducing sensor to shooter times. To that end, this thesis is organized in the following manner:

The Introduction chapter, Chapter I, sets the foundation for the SWEB and why it is important to the military. It briefly surveys the vision and component technologies of the SWEB, and then presents two hypotheses about the SWEB. To continue we examine our version of the Observe, Orient, Decide, Act (OODA) Loop modeled using a Causal Loop diagram (CLD) from the theories of Systems Dynamics (SD). The CLD will indicate (causal) relationships between elements of the process. We highlight the bottlenecks and delays of the current decision making system, and we identify the potential leverage points for process improvement. These leverage points are the areas we will target for potential application of SWEB technologies. The remaining chapters are dedicated to describing these technologies and demonstrating their strengths, weaknesses and functionality through examples and technical analysis.

The Theory chapter, Chapter II, provides a "forward look" through the processes contributing to the promised output of the Semantic Web (SWEB); that is, the construction of knowledge. Through the principles, theories and concepts presented in this chapter we establish a premise from which to achieve a common frame of reference of the concept knowledge as it relates to the SWEB. The common frame of reference

will serve as the starting point for our discussion regarding the complexities, dynamics and challenges an integrated SWEB application/system must overcome to achieve its goal. It will also assist in understanding the contributions each SWEB technology/component analyzed in this work, and how each helps to construct knowledge from enabled network content.

Chapter III, Data Sources, serves to underscore the importance of data sources residing on a network to the SWEB's knowledge generation process. We discuss various types of data sources likely encountered and methods to successfully enable them. The implications of structured data, and how the Extensible Markup Language (XML)[1] relates to different database models are key points of this chapter. Another point this chapter develops is the importance of representing data in such a way that it becomes machine-interpretable information. With machine-readable information present we are able to develop agents capable of automatically, autonomously understanding the information and responding appropriately. However, prior to discussing agents, we discuss the methods agents will employ to access and "read" this stored information. These methods may be classified under the umbrella of "Distributed Computing." Effective distributed computing technologies and techniques factor importantly in connecting "small worlds" of information repositories.

The Distributed Computing chapter, Chapter IV, demonstrates an effective means to interact with and across the network. Our emphasis will be on how the present and potential distributed computing mechanisms can best help to enable the SWEB. The progression of distributed computing models has seen a number of designs, and it fulfills a vital role in a new distributed computing paradigm called "Web services." We believe Web services, more precisely, SWEB services will prove integral to improving workflow between organizations. To understand the SWEB services' role it is important to survey the progression of distributed computing. As such, we highlight some of the more prevalent distributed computing models including Web services, which are presently

[1] This paper assumes a basic working knowledge of XML. Many sources are available to learn XML including [http://www.w3c.org/xml].

achieving much notoriety in the distributed computing world. We then discuss SWEB services, an extension of Web services. The SWEB services model will be analyzed for utility and applicability to the SWEB, especially for military purposes.

Chapter V, Agents, discusses the notion of software agents, their relationships to each other and to humans, and considerations for their employment in military operations. Additionally, this chapter discusses a sampling of the agents we developed in support of an agent-based prototype application called “ArchAngel.” The purpose of this chapter is two-fold. That is, it provides an abstract conceptual underpinning for agents, and follows with concrete illustrations.

The Ontology chapter, Chapter VI, highlights the importance of the ontology to the SWEB. We review the Web Ontology Language (OWL) and analyze the basic components of an ontology. We highlight ontology design criteria, methodologies and various ontology design patterns. Our discussion will culminate by exposing several of the challenges a developer will face while designing and deploying an ontology for practical use.

The purpose of Chapter VII, SWEB Knowledge Base, is to demonstrate the importance of the network to the Knowledge Base of a Semantic Web (SWEB) application and its critical components. We review design patterns and considerations contrasting the Knowledge Base (KB) of an SWEB application with the KBs supporting the Expert Systems (ES) of the 1990’s. We demonstrate how the SWEB KB can reason against enabled content and discuss techniques for design and organization. This chapter is intended to guide the reader in attaining a better understanding of the functions and interactions of a traditional KB, as well as a “networked” KB, as it will likely occur in SWEB applications. We will discuss the KB in the traditional AI terms from its definition, design criteria, components, organization. This discussion will rely on the recollection of many of the foundational concepts we have discussed in early chapters as the KB is where all the concepts converge.

In our concluding chapter, Chapter VII, we offer a refined hypothesis describing our vision of the transition to the Semantic Web. We identify six key leverage points within the Causal Loop Diagram wherein SWEB technologies will help to reduce decision cycle times, and decrease uncertainty.

I. INTRODUCTION

> The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.
>
> -- Tim Berners-Lee, James Hendler, Ora Lassila, *The Semantic Web*, Scientific American, May 2001

A. PROBLEM STATEMENT

Recent military operations have redefined the way modern warfare is waged. In a deliberate effort to achieve and retain information dominance and decision superiority, many innovative technologies have emerged to assist the human war fighter. Unquestionably, these technologies have generated resounding successes on the battlefield, the likes of which have never been seen. The speed, precision, and accuracy involved in the process of evaluating data, applying knowledge, generating decisions, and ultimately carrying out effective strikes are the foundation for these successes. With all the success, however, there are still areas for improvement as the potential exists for reducing already short sensor-to-shooter times further. The subject of this study was to evaluate the technologies of the Semantic Web (SWEB) for potential application to Military Operations. The goal of this effort was to assess potential technology insertion points into processes within Military Operations with the potential to further assist the decision maker in making more efficient and effective decisions; where efficient equates to minimum time, and effective equates to the correct decision, all supported by the most relevant intelligence. For this work our surrogate for intelligence is knowledge, and the difference between knowledge and information is that knowledge is actionable. The generation of actionable knowledge is the ultimate goal of the SWEB.

B. SURVEY OF THE SEMANTIC WEB (SWEB)

1. The Vision

The World Wide Web Consortium (W3C), the proponent of the SWEB, defines the SWEB as the representation of data on the World Wide Web (W3C Semantic Web, 2001). The current World Wide Web (WWW) is largely a human centric information space where humans exchange and interpret data ([2] Berners-Lee, 1, 1999). The SWEB is not a separate Web, but an extension of the current one in which content is given well-defined meaning, better enabling computers and people to work in cooperation (Berners-Lee et al). In the SWEB information space, the content residing in documents, portions of documents, or other mediums is described by explicit relationships between the entities/concepts of the domain creating machine interpretable content ([2] Berners-Lee, 2, 1999). The SWEB then connects the machine interpretable content available from the distributed, independent contributing communities forming the Web of understanding or the SWEB ([2] Berners-Lee, 4, 1999). The result is the availability of the various backgrounds, experiences, and abilities of the contributing communities through the self-describing content populating the SWEB ([2] Berners-Lee, 1999). The availability of such content allows for efficient aggregation, from which machine interpretable knowledge and understanding can ultimately be constructed (Daconta et al., 17, 2003). The construction of knowledge from disparate, raw data is the ultimate promise the technologies of the SWEB intend to deliver. The potential of realizing this promise in the military domain must be pursued.

2. Components of the Semantic Web (SWEB)

The SWEB is composed of three basic components, Knowledge Representation (KR) languages, ontologies, and logic. Although not a primary SWEB component, agent technology is an important beneficiary of the SWEB environment worthy of mention in this section. While all components currently operate independently on the WWW of today, the real power of the SWEB will not be realized until all are operating seamlessly and synergistically in concert. To ensure familiarity with the concepts of the SWEB we will discuss each of the primary components of the SWEB including agents, due to the potential value they will add. This section will serve to establish the background required to gain a basic understanding of the SWEB for the purposes of understanding this work as it will be applied in the military domain.

a. Standardized Knowledge Representation (KR) Language

Standardization of KR languages is a significant step in ensuring the SWEB will become reality. Such languages rely on abstracting logic in user friendly syntax to express and represent concepts in machine interpretable form. The Extensible Markup Language (XML) and Resource Definition Framework (RDF), now well established with significant implementation experience, provide the baseline for the emerging Web Ontology Language (OWL), the impending W3C Recommendation for a standard KR language. OWL provides the abstract syntax enabling content to be tagged with semantic meaning by describing it relative to other described entities/concepts within the domain in the form of triples (subject, predicates, and objects) and establishing relationships between them. OWL further describes the triples and their relationships by assigning a Uniform Resource Identifier (URI), or name space, allowing the content to assert in machine interpretable syntax (logic) how a given triple is related to other triples. The functionality and implementation examples of OWL will be covered in detail later in this work. Once OWL is established, the concept of ontologies must be formalized to ensure described content is unambiguously interpreted by capable machines.

b. Ontologies

Ontologies, or "the theory about the state of existence in a domain," provide an unambiguous, machine interpretable solution ensuring entities are interpreted as their meaning was intended. Ontologies for specific domains establish classes, properties and relationships governing the content by a machine interpretable, logic based specification expressed by the ontology. Not only does the ontology establish the specification for individuals, classes and properties, but they also establish how entities in the domain described within the ontology relate. Ontologies are a critical component of the SWEB and because of this we devote an entire chapter to its study.

c. Logic

A portion of the logic component of the SWEB is partially embedded within the ontology. Additional logic can be applied by incorporating external, situational or event driven rules. The logic embedded within the standardized KR language and the ontology, combined with the optional external rules, form the domain theory. The domain theory governs the actions of all activities within the domain. Logic enables machines and software agents the ability to reason against the described content to answer questions, classify concepts and fire rules triggering action. Once this component is established machines can negotiate, render conclusions based on an interpreted set of facts, or optimize functions based on constraints. These actions will be accomplished with little or no human interaction, freeing the human to focus on the tasks requiring higher order reasoning. Logic and reasoning will be discussed in detail in later chapters.

d. Agents

The maturation and adoption of agent technology will change the roles of computers as we know them. Agents, empowered by the semantically rich environment will be used to accomplish the mundane, repetitive, and time consuming tasks currently occupying the majority of human computing time. For example, E-commerce transactions will rise to a new level of efficiency as mobile, autonomous agents transit between vendors to accomplish Business-to-Business (B2B) or Business-to-Consumer (B2C) tasks on behalf of human actors. Agents can continue their tasks indefinitely as they have unlimited endurance, yet still require a human user to input parameters regulating their actions and behaviors. Agents will be capable of communicating with other agents to pass value added information forming systems of agents. Agent technology is currently immature, but as the SWEB expands, agent technology will be sure to follow.

C. MOTIVATION

The war fighter combats uncertainty by collecting, analyzing and ultimately acting on knowledge. The intelligence collection platforms of today are capable of generating incalculable amounts of raw data. These data are required to be analyzed, prioritized and disseminated by humans interfacing with networks. Because of the reliance on the human, even for the menial tasks such as classifying, comparing and correlating, we are subject to the limitations of the human cognitive capacity. Today's operations, even with the application of Information Technology (IT), reveal an alarming amount of unread message traffic, unused sensor data, and uncorrelated facts, largely attributed to data overload. As the proliferation of sensors and bandwidth increases, the quantity of available data increases, thus imposing a time and manpower overhead to interpret the data for decision-makers. More and more aspects of war-fighting are not only leaving the realm of human senses, but crossing outside the limits of human reaction times (Adams, 2001, 58). Furthermore, the proliferation of information-based systems will produce substantially greater data overload eventually making it impossible for humans to absorb and discern the necessary information/knowledge value from the mounting data glut. The results are decisions based on incomplete information. The technologies of the SWEB allow machines to assist humans in interpreting the data glut and incorporating it in the construction of domain knowledge. The use of machine interpretable data is the key to constructing knowledge, knowledge is the key to "battlefield dominance," and the speed at which we act on knowledge is the key to battlefield success.

Because organizations, individuals and the military desire the highest possible degree of certainty from which to make decisions, they have pursued knowledge wherever it was perceived to exist. While this pursuit of knowledge is sound and justified theoretically, it is extremely difficult, time consuming and complex to realize in practice with the current WWW technologies. The SWEB and its supporting technologies will assist in the implementation of the above paradigm by abstracting the complexities and addressing the difficulties associated with data aggregation and knowledge representation, all of which is ultimately used in the construction of knowledge. The ability to leverage machine interpretable data to construct knowledge and the interpretation of that knowledge by machines will further assist military decision makers in achieving a higher degree of predictability within a given problem space than was previously possible - all with decreased human intervention.

1. Hypotheses

Our research exposed two competing hypotheses associated with the widespread adoption of the SWEB. Hypothesis A argues the transition to the SWEB will be swift, disruptive and revolutionary with a degree of payoff for early adopters. Conversely, Hypothesis B argues a gradual transition or evolution to the SWEB. It sees little incentive to the early adopter as enterprises realize the full potential of the SWEB cannot be reached until a significant number of adopters proliferate. Both complete hypotheses can be found in Table 1.

Hypothesis A	***Hypothesis B***
The application of SWEB technologies including ontologies, agents, knowledge networks and reasoning will revolutionize the current WWW and be adopted quickly. The relative speed of adoption will stimulate tool and methodology development, as well as the necessary cultural changes. It will therefore allow the technologies of the SWEB to be adopted relatively swiftly, with considerable disruption equivalent to the disruptions produced by a revolution.	SWEB technologies have limitations and constraints. Hype has created the illusion the SWEB is *the* solution to the problems of data overload and information/knowledge management. The barriers and adoption inhibitors may make the SWEB Revolution more of a gradual evolution unrealized by users of the current WWW. The current barriers to entry as well as adoption inhibitors include technology, culture, training/education, tools, and above all, realizable and quantifiable, short term Return On Investment (ROI) or Knowledge Value Added (KVA). The slow payoff from the SWEB investment will likely curtail the largely academic implementation momentum and subdue any incentives or competitive advantage early adopters can achieve.

Table 1. Research Hypotheses.

2. Model

To lend rigor to our hypotheses and to ground it in a familiar concept, we elected to map our model to the well known Observe, Orient, Decide and Act (OODA) Loop invented by Colonel John Boyd, United States Air Force (Ret.) (See Figure 1). Figure 1 overlays the traditional OODA Loop model on to our model, a Causal Loop Diagram (CLD). The OODA Loop theory is based on the fact that to be successful in warfare an individual must continuously Observe, Orient, Decide and Act, culminating in a decision. The more rapidly an individual/system can cycle the OODA loop, the more rapidly one can make sound, decisions. The time it takes to cycle the OODA Loop is called cycle time (Boyd, 1970's).

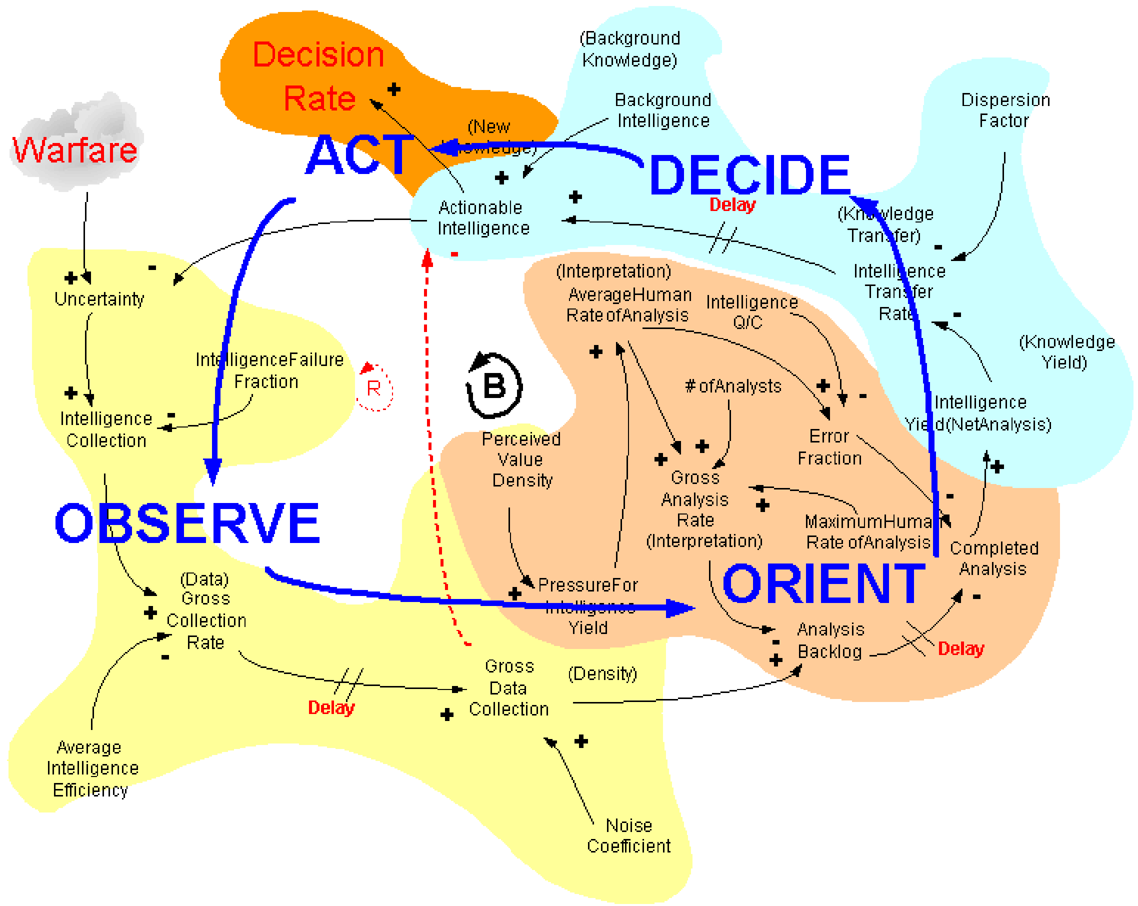


Figure 1. Causal Loop Diagram Mapped to OODA Loop.

To continue we will examine our version of the OODA Loop modeled using a CLD from the theories of Systems Dynamics (SD). The CLD will indicate (causal) relationships between elements of the process. We highlight the bottlenecks and delays of the current decision making system, and we identify the potential leverage points for process improvement. These leverage points are the areas we will target for potential application of SWEB technologies. The remainder of this work is dedicated to describing these technologies and demonstrating their strengths, weaknesses and functionality through examples and technical analysis.

3. Leverage Points

With our model grounded in the concept of the OODA Loop, we intend to generate a degree of familiarity to achieve a better understanding of our model, as they are essentially the same. Our model depicted in Figure 2 highlights the bottlenecks and inefficiencies of a typical deliberate military decision making process[2]. These are the areas we will keep in mind when discussing the SWEB Technologies. Since speed is a key aspect of decision making, many of the potential leverage points are located at positions of naturally occurring delays. The other leverage points we have identified are located at points of human inefficiencies or points where error can be introduced in the decision making process. The leverage points are identified in Figure 2 by numbered red dots and are further elaborated in the discussion that follows.

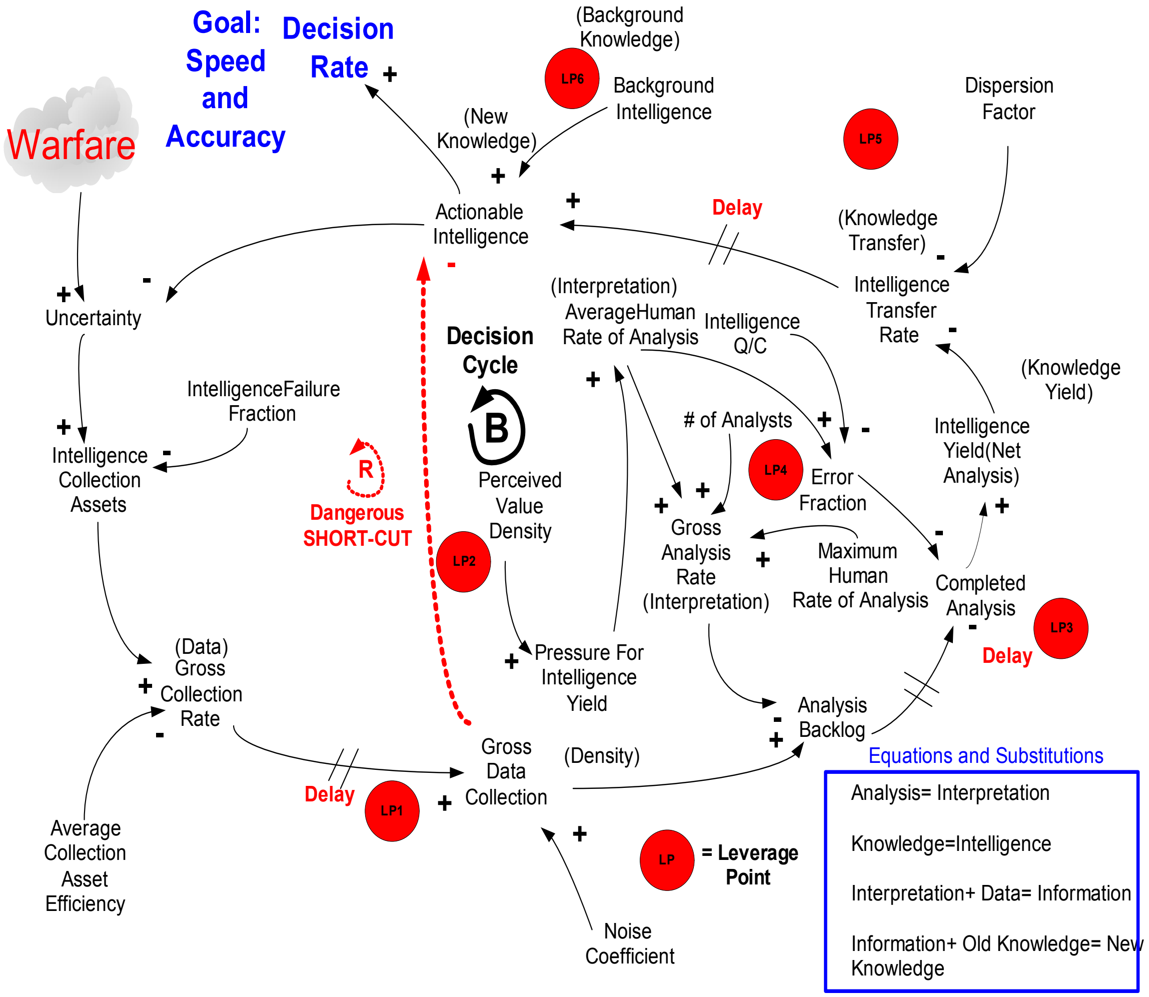


Figure 2. Military Decision Making Causal Loop Diagram and Potential Leverage Points.

[2] The rapid military decision making process is also appropriately described by this model.

4. Model Analysis

a. Feedback Loop

To ensure our model is understood we will step through the elements by OODA Phase. We will communicate the meaning of our model in terms of SD Theory. For readers unfamiliar with SD, the notation in the diagram includes arcs connecting the variables of the model. The arcs represent causal links between variables. Each causal link is assigned a polarity, either positive (+) or negative (-), to indicate how the dependent variable changes as the independent variable changes. Important loops are named and highlighted by a loop indicator. The two significant loops in our model are Dangerous Shortcut, a reinforcing (R) or growth loop, and Decision Making Cycle a balancing (B) or goal seeking loop.

b. Polarity Indicators

The polarity indicators on the arcs are interpreted as follows. A positive (+) link means if the cause increases, then the dependent variable increases beyond what it would have otherwise. Additionally, a positive causal link means if the causal variable decreases, then the dependent variable decreases below what it would be otherwise. Conversely, a negative causal link means an increase in the casual variable results in a decrease in the dependent variable below what it would have otherwise; and a decrease in the causal variable leads to an increase in the dependent variable above what it would be otherwise. The rest of the model is interpreted by reading the text descriptors. There will be noticeable overlap between the phases. The overlap is important, because it can result in a boundary condition having its own special set of dynamics. As you read the text description we recommend following along with the model in Figure 2.

c. Observe Phase

This portion of the model starts at Warfare and continues to Perceived Value Density. The following is an interpretation of the Observe Phase of our model:

- Warfare leads to increased Uncertainty.
- Increased Uncertainty leads to increases in the deployment of Intelligence Collection Assets

- Intelligence Failure Fraction has a negative effect on Intelligence Collection Assets
- More Intelligence Collection Assets result in a higher Gross Data Collection Rate
- Average Collection Asset Efficiency has a decreasing effect on Gross Data Collection Rate. This occurs because the asset can either be operating at 100% efficiency, meaning it is collecting at its target collection rate or some value less than 100% such as 70% in which it is collecting below its target collection rate. Therefore, if Average Collection Asset Efficiency is below 100% the Gross Data Collection rate will be negatively affected.
- An increasing Gross Data Collection Rate, after some Delay, leads to more Gross Data Collection
- Noise Coefficient has a positive effect on Gross Data Collection. That is to say that more noise leads to more Gross Data Collection as Gross Data includes "good" and "bad" data
- (Dangerous Short-cut Arc) If exercised, the Gross Data Collection can have a negative effect on Actionable Intelligence. See LP 2 below.
- Perceived Value Density has a positive effect on Pressure for Intelligence Yield. The more active we perceive the collection environment, the more we desire to collect, often disregarding our system's capacity to process/analyze the results

(1) Leverage Point 1 (LP1). Leverage Point 1 is identified between the interaction of Data Collection Rate and Gross Data Collection. The delay depicts the Collection Delay originating from the collection of undescribed, potentially unorganized data without meaning from the collection assets. Resources must be allocated from the system to sort, analyze and disseminate the data to the analyst(s) responsible for interpreting the data. Additionally, the Noise Coefficient can interject valueless background clutter into Gross Data Collection. If the Noise Coefficient is high, it can tremendously increase the amount of gross data collected, causing additional resource allocation to the processing effort occurring before interpretation or analysis. This delay, the focus of this leverage point, can be substantial and is directly related to the amount of data being collected.

(2) Leverage Point 2 (LP2). Leverage Point 2 is the dynamics resulting from the Perceived Value Density of the Decision Maker and the Pressure for

Intelligence Yield imposed by the system as a consequence. If the Pressure for Intelligence Yield gets too great, there may be temptation to take a dangerous shortcut. The shortcut occurs when data is forwarded directly from the Gross Data Collection to the Decision Maker through Actionable Intelligence. If this occurs Gross Data Collection has a negative effect on Actionable Intelligence because it interjects unevaluated, raw data into the Actionable Intelligence (New Knowledge) the Decision Maker is basing decisions. The results of this dynamic can slow the Net Decision Rate resulting from the pollution of Actionable Intelligence by raw unevaluated data.

d. Orient Phase

The Orient Phase begins with Analysis Backlog and ends with Intelligence Yield. The following is an interpretation of the Orient Phase of our model:

- Greater (or less) Gross Data Collection increases (or decreases) Analysis Backlog
- Perceived Value Density has a positive effect on Pressure for Intelligence Yield
- Increased Pressure for Intelligence Yield tends to increase the Average Human Rate of Analysis with penalty
- Average rate of Human Analysis has a positive effect on both the Gross Analysis Rate (Interpretation) and Error Fraction
- Increased Gross Analysis Rate (Interpretation) decreases the Analysis Backlog
- Maximum Human Rate of Analysis (MHRA) has a positive impact on Gross Analysis Rate
- More Intelligence Quality Control (Q/C) reduces Error Fraction
- Analysis Backlog with Analysis Delay has a negative effect on Completed Analysis
- Error Fraction has a negative effect on Completed Analysis
- Completed Analysis has a positive effect on Intelligence Yield (Net Analysis)

(1) Leverage Point 3 (LP3). LP3 is targeted at the Analysis Delay largely caused by the Maximum Human Rate of Analysis (MHRA). The MHRA is a result of the human cognitive limits.

(2) Leverage Point 4 (LP4). LP4 is targeted at a potential, unanticipated side effect from MHRA and the effects of Pressure for Intelligence Yield. The more rapidly a human analyzes material the greater the fatigue factor. The more fatigue, the larger the Error Fraction. The larger Error Fraction does not result in Completed Analysis which is directly counter to our process goal. The most important point from LP4 is the fact that the human can analyze, at a certain rate, for a certain amount of time. After a point, fatigue, errors and the temptation for abandoning the process and taking short cuts increase and lead to short-cuts to and abandonment of the process. The resulting positive reinforcing feedback loop ultimately causes a slower decrease in Uncertainty.

e. Decide Phase

The Decide Phase begins with Intelligence Yield and ends with Actionable Intelligence. The following is an interpretation of the Decide Phase of our model:

- Intelligence Yield has a negative effect on Intelligence Transfer Rate
- Increasing Dispersion also decreases Intelligence Transfer Rate
- Greater Intelligence Transfer Rate, after a delay, leads to more Actionable Intelligence
- Background Intelligence also has a positive effect on Actionable Intelligence

(1) Leverage Point 5 (LP5). LP5 focuses on the transfer delay in the Intelligence Transfer Rate between Intelligence Yield and Actionable Intelligence. The delay is caused by latency in the network, network traffic (bottlenecks/collisions) and network availability. Additionally, more Intelligence Yield effectively reduces the Intelligence Transfer Rate as more Intelligence is available but the intrinsic Intelligence Transfer Rate is unchanged. Finally, the intrinsic Dispersion Factor of knowledge, in this case our surrogate Intelligence, also affects this leverage point. The higher the Dispersion Factor the greater the adverse effects on the system’s ability to transfer intelligence (Intelligence Transfer Rate) to its intended recipient.

f. Act Phase

The Act Phase begins with Actionable Intelligence and ends by closing the balancing or goal seeking feedback loop at Uncertainty. The following is an interpretation of the Act Phase of our model:

- Actionable Intelligence has a positive effect on Decision Rate
- Actionable Intelligence has a negative effect on Uncertainty, closing the balancing or goal seeking feedback loop

(1) Leverage Point 6 (LP6). LP6 is found in the added value of relevant, usable and available Background Intelligence (Background Knowledge) to create new Actionable Intelligence (New Knowledge). This dynamic is better explained by the New Knowledge Equation: Old Knowledge + Information = New Knowledge. From this equation we gain a better appreciation for the importance of Background Intelligence to the process. This equation will be further explained later in the work.

D. RESEARCH EFFORTS

Our research approach primarily consisted of studying the theoretical concepts within the SWEB domain, developing the theories through the application of SWEB technologies in military examples and documenting the results. This process essentially repeated itself throughout our research. We began by studying XML and realizing its enormous contributions to the SWEB through its flexibility of use. We explored the application of the Web Ontology Language (OWL) and developed a series of ontologies exposing valuable design patterns. We studied collaborative technologies with an emphasis on software agents developing multiple agents using the Control of Agent-Based Systems (CoABS) platform. We implemented a working exemplar involving software agents interacting with data sources available on the network. We subsequently investigated the concept of a Networked Knowledge Base and how it relates to an integrated SWEB application. Additionally, we implemented forward chaining computer reasoning exemplars using ontologies to apply the domain theory. We made significant inroads to understanding the implementation challenges of SWEB technologies. As we illustrate in the Future Work section, the integration of the SWEB technologies at the leverage points we have identified in a working application is a logical next step.

E. THESIS ORGANIZATION

The chapters in this thesis discuss the technologies and technology areas central to the advent of the SWEB. To that end, this thesis is organized in the following manner: Chapter II explores theories related to forming a common frame of reference when discussing the SWEB technologies and how they contribute to the construction of knowledge, the importance of machine interpretable knowledge to the SWEB, the different characteristics of knowledge and different knowledge types. Chapter III describes the importance of networked data sources, characteristics and comparison of Relational Databases (RDB) and Native Extensible Markup Language Databases (NXD), and methods for enabling inclusion of a data source to drive a SWEB application. Chapter IV discusses the necessity and importance of distributed computing, an overview of distributed computing technologies, an overview of web services and SWEB services, and implementation options. Chapter V outlines the emerging technology of software agents, agent categories, outlines application areas, describes adoption inhibitors and challenges, and illustrates the concepts with annotated, working examples. Chapter VI overviews ontologies, the Web Ontology Language (OWL), knowledge representation (KR) concepts, ontology design criteria and methodology, classification of ontologies, and illustrates design patterns grounded in validated OWL examples. Chapter VII analyzes the concept of the roles and functions of a Knowledge Base (KB) in a SWEB application, defines KB, details the components of a KB, demonstrates reasoning and the application of an external, rule based system through examples, and describes design and organization criteria for a KB.

F. ONWARD TO THE SWEB

Now that we have established a common frame of reference and communicated the required background information to proceed with our analysis, we must understand our efforts will focus the SWEB technologies at the leverage points we identified in our model. To further ground our analysis let us familiarize ourselves with some of the underlying theories, concepts and principles critical to the understanding of how SWEB technologies will function in their endeavor to construct knowledge by enabling the content sources of the WWW.

II. THEORY

Knowledge must become capability

-Carl Von Clausewitz

A. BACKGROUND

This chapter provides a "forward look" through the processes contributing to the promised output of the Semantic Web (SWEB); that is, the construction of knowledge. Through the principles, theories and concepts presented in this chapter we establish a premise from which to achieve a common frame of reference of knowledge as it relates to the SWEB. Many of these concepts are intentionally abstracted away from the user/developer, but the importance of gaining familiarity with the foundational concepts and underpinnings of the SWEB technologies we will be discussing remains. The common frame of reference we establish in this chapter will serve as the starting point for our discussion regarding the complexities, dynamics and challenges an integrated SWEB application/system must overcome to achieve its goal. Many of these challenges originate in the foundational concepts. This chapter will also assist in understanding the contributions each SWEB technology/component analyzed in this work makes toward the effort of constructing knowledge from enabled network content.

"War is the realm of uncertainty; three quarters of the factors on which actions in war are based are wrapped in a fog of greater or lesser uncertainty" (Von Clauswitz, 1976, 101). As such, the most popular remedy to counter uncertainty is knowledge (Davenport, 2000, 25). Therefore, if one possesses "complete and accurate knowledge" regarding the outcome of a pending decision, one possesses total certainty (Marakas, 1998, 60). While the probability of achieving total certainty in warfare is close to zero, the probability of complete uncertainty is also close to zero (Marakas, 1998, 60). It is within these bounds of the decision making continuum the SWEB aims to construct computer interpretable knowledge from aggregations of enabled content to assist the military decision maker, ushering in a quiet paradigm shift from a data centric force to a fighting force based on knowledge. Ultimately, the technologies of the SWEB have the potential to create greater computer assisted efficiencies within the networked communities of war fighting.

As with most emerging technology initiatives, the Department of Defense (DoD) has been observing and evaluating the paradigm shift within industry, research communities and academia looking for opportunity. In recent years DoD has recognized the potential value added of knowledge driven applications and has focused significant effort into realizing the transformation. Joint Vision 2020, the Department of Defense's guide to the transformation of America's Armed Forces, outlines the requirements for the 21st Century Joint Force to take advantage of superior information *converted* to superior knowledge to achieve "decision superiority" (Joint Vision 2020, 2000, 11-12). While Joint Vision 2020 mandates the transformation to a knowledge based force, it also explicitly acknowledges requirements for conversions and transformations of data/information to generate the relevant knowledge to be applied to the decision, which will ultimately lead to "decision superiority". How will these prescribed conversions occur? What needs to be converted? How will this knowledge be generated? The SWEB can provide us with part of the answer.

1. Knowledge Warrior (Tofler, 1993, 140)

Alvin and Heidi Toffler, renowned futurists, spoke of "knowledge warfare" as the most influential factor of the Third Wave form in their 1993 work War and Anti-War (Toffler, 1993, 139). Clausewitz and Sun Tzu regarded knowledge as critical to battlefield success. Prosecuting a military campaign requires an enormous amount of knowledge to succeed. Therefore, the realization of the SWEB, a technology built around knowledge generation and application should be of great interest to the Department of Defense. In fact the Department of Defense through the Defense Advanced Research Projects Agency (DARPA) has sponsored numerous research efforts focused on such realization[3]. The SWEB promises to assist users with the discovery, generation, storage, transfer, maintenance and reuse of knowledge, and cannot be ignored for its potential applicability to DoD. These activities, once realized and mature, could

[3] DARPA Programs found at [www.darpa.mil/body/darpaoff.html], 10 July 2003.

have the potential to assist military decision makers by supplying them with the right knowledge at the right time (Davies, 2003, 2-5), increasing decision rates, and qualitatively and quantitatively improving decisions. To understand the aspects of knowledge the SWEB technologies can leverage innovation against, it is necessary to form a working definition, and more importantly a common frame of reference for the meaning of knowledge.

2. Semantic Theories and Knowledge Views

Exposure to the different semantic theories or knowledge views is important to discuss before we can establish the relationship of knowledge to the SWEB. If we understand the semantic theories we can gain insight to how a domain is attempting to achieve meaning. Semantic theories determine the view one will take to achieve interpretation. Our definition will invoke the intensional semantic theory as our concept of knowledge will be mapped to a set of possible worlds within the SWEB domain. It is important to note that within a given knowledge application we can implement more than one semantic theory. We will discuss the extensional and intensional semantic theories in this section.

a. Extensional Semantic Theory

The extensional semantic theory is perhaps the most intuitive of the semantic theories. Extensional knowledge is that knowledge specific to a particular problem or a set of individuals. The extensional knowledge is the A-box or assertional knowledge about a domain. It is knowledge that can be thought of as a term’s denotation (Sowa, 2000, 99), or the class of objects which a set refers. In extensional semantics, the constituents of the language become mapped onto a “world” model (Gardenfors, 2000, 152). These mappings are then formed into sentences mapped to truth values. These sentences then formulate the truth conditions for this “one world” (Gardenfors, 2000, 152). Knowledge in this category tends to be more dynamic and may require maintenance and refresh to retain relevancy. An extensional definition for military helicopters would be a catalog of all military helicopters in the world (Sowa, 2000, 99). Ontologies capturing entities with regards to their existence can generally be classified as having an extensional knowledge basis (See Figure 3).

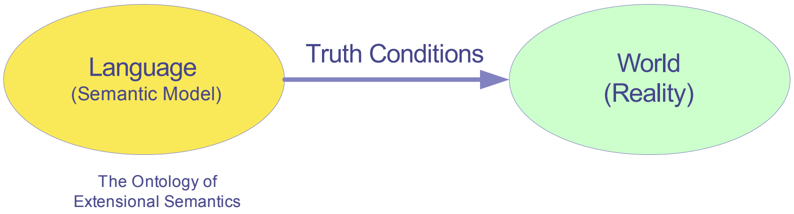


Figure 3. The Ontology of Extensional Semantics (After: Gardenfors, 2000, 153).

b. Intensional Semantic Theory

The intensional semantic theory establishes truth conditions for a set of "possible worlds". The intension of a term means its intrinsic meaning or associated concepts (Sowa, 2000, 99). Intensional knowledge is described as general properties of the concepts within a domain. This is the T-Box knowledge, or term knowledge. Within this theory there are multiple circumstances in which a given condition can be considered true. Put another way, besides the true state of affairs there are a number of other possible "worlds" (Fagin, 1995, 15). An intensional ontology for military helicopters would specify properties or criteria for identifying military helicopters without regard to their possible existence (Sowa, 2000, 99). For instance every helicopter has one or more main rotors, has one or more engines, has at most one tail rotor and is flown by one or more pilots. An enumeration of all parts of a helicopter establishes the truth conditions for all helicopters in our domain of discourse; therefore, it becomes our "possible worlds". The preconditions to incorporate this semantic theory are that the truth conditions of the "worlds" are well understood and somewhat static and predicable. Extensional knowledge tends not to change rapidly retaining its utility and relevance for relatively long amounts of time (See Figure 4).

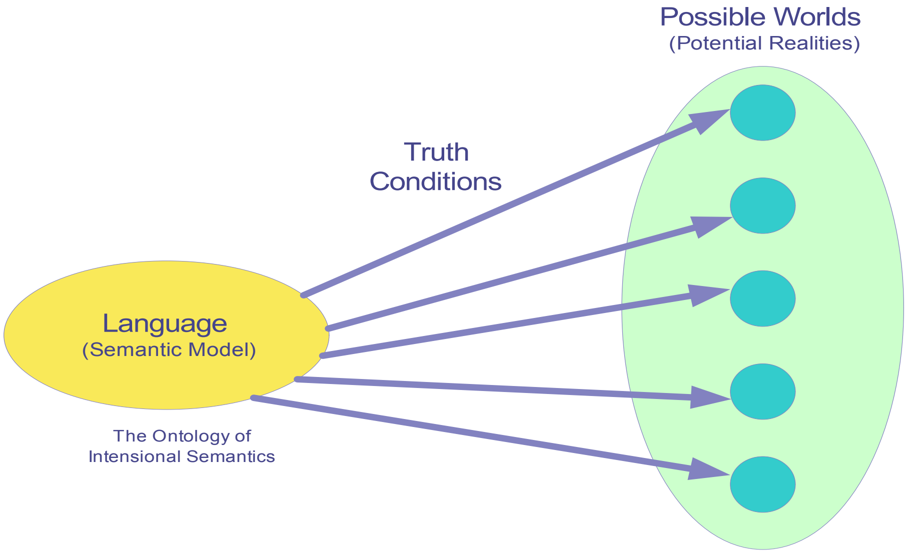


Figure 4. The Ontology of Intensional Semantics (After: Gardenfors, 2000, 153).

Now we have been exposed to the general semantic theories and can distinguish between the two types, let us continue our analysis to establish our common frame of reference and meaning of knowledge as it relates to the SWEB.

B. THE MEANING OF KNOWLEDGE

The concept of knowledge is very difficult to define, as are most products of the human mind, including software and intellectual property. Most people when asked to provide a definition of knowledge could provide a listing of attributes and terms to describe knowledge based on their own perceptions, but most would have a difficult time deciding on an acceptable definition. This is understandable, as specifying the definition, nature and contents of "knowledge" even for experts can be daunting (Housel, 2001, 1). According to I. A. Richards and C. K. Ogden's famous study on the influence of language upon thought, "All definitions are essentially a*d hoc*." "They are relevant to some purpose or situation and consequently are applicable only over a restricted field or 'universe of discourse'" (Ogden, 1923, 111).

Ogden and Richards's prescription to this problem requires finding a common set of referents about which agreement can be secured and locating the required referent through its connection with these (Ogden, 1923, 113). A referent can be defined according to the Oxford English Dictionary as the "object of a reference". With this in mind, we will dissect many of the common definitions of knowledge with the purpose of distilling a set of attributes to connect to knowledge. Next we will analyze this set of attributes and their associated implementation challenges and examine how the SWEB and its supporting technologies will effectively cope with this expected range of difficulties associated with constructing knowledge into a machine interpretable form. Through this analysis we will establish a common meaning of knowledge for the purpose of gaining an understanding of the promised goal of the SWEB.

Fortunately, epistemology, the study of "what knowledge is," has supplied us with many definitions[4] of knowledge, all derived for different purposes. We will use these available definitions of knowledge as data input to our analysis to construct a meaning triangle applicable to the concept of knowledge as it applies to the SWEB. The following enumeration of knowledge characteristics found below has been extracted from numerous definitions of knowledge and will be subject of our analysis. These characteristics were chosen for their frequency of occurrence among many of the different knowledge definitions and their potential value and challenges they could impose on the realization of the SWEB. As the analysis proceeds the characteristics listed below will be decomposed as required to permit discovery of other influential characteristics couched within. Figure 5, the Knowledge Meaning Triangle, illustrates our points in graphic form. The one-to-one relationship established by the links between each pair of terms hides the inherent complexity of a one-to-many relationship (Maedche, 2002, 14). The links can only be completed when the interpreter processes the term and invokes the corresponding concept; then links the concept to the term referent in the real world (Maedche, 2002, 15). Readers should not consider the below enumeration to be exhaustive.

[4] We used elements of the following expert's definition of knowledge: Sowa, Davenport, Hayek, Orbst, McGuinness, Klein, Fensel.

<u>Knowledge Characteristics</u>

- Is Experience
- Complexity
- Requires Interpretation
- Embeds in everything
- Has Shape
- Transferable
- Actionable

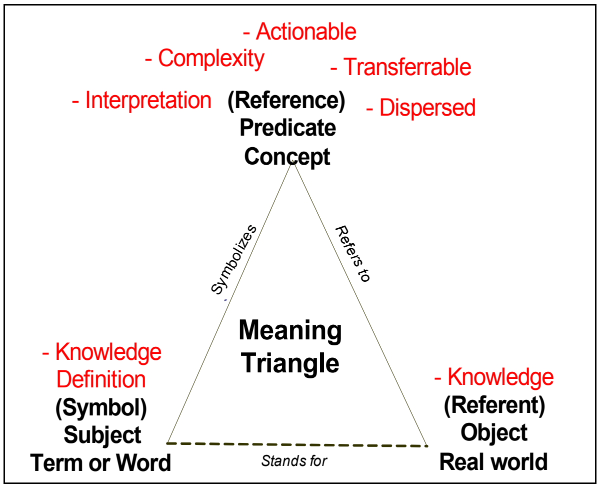


Figure 5. Knowledge Meaning Triangle (After: Ogden, 1923, 11).

The Meaning Triangle begins at the lower, left vertex with our stated goal, ‘Knowledge Definition’. The ‘Knowledge Definition’ becomes our symbol or subject with which we will associate additional ‘Thoughts’ or objects from our dissection of a representative set of published knowledge definitions. We then proceed up the left leg of the triangle, the ascending triple, to the apex. The property or predicate ‘Symbolizes’ from the ascending leg becomes the connection between the lower left vertex and the apex. We read the ascending triple as:

“Knowledge Definition Symbolizes (Substitute a Reference Term from the Apex) Complexity.”

Next we proceed down the right hand leg of the meaning triangle, using the predicate ‘Refers to’, connecting the apex with the lower right vertex or our ‘Referent’, ‘Knowledge’. We read the descending triple as: “Complexity Refers to Knowledge." Putting both the ascending and descending triples together we establish:

“Knowledge Definition is Symbolized by Complexity which Refers to Knowledge.”

Next we are able to connect the lower left vertex with the lower right vertex utilizing the base of the triangle and the predicate ‘Stands for’. We read this as:

“Knowledge Definition Stands for Knowledge.”

By executing this connection exercise with multiple references we effectively connect our 'Knowledge Definition' with 'Knowledge' through the attribute set derived from the different knowledge definitions on the apex. Each of these connections now associates meaning through the predicate and establishes our common meaning of knowledge as it applies to the SWEB. Let us analyze each of these 'References' now associating our Knowledge Definition with the concept of Knowledge to test the validity of the associations.

1. Knowledge Characteristic Analysis

a. Complexity

The complex nature of knowledge provides one of the most formidable challenges to the adoptions of the Semantics Web. John Sowa describes knowledge in its various forms as "Knowledge Soup" due to its inherent complexity and disorganization (Sowa, 2000, 348). The idea of knowledge as a complex entity is shared by many leading knowledge researchers (Orbst, 2003, 104) (Davenport, 2000, 9) (Riedl, 2002, 45). The complex nature of knowledge should not come as a surprise as knowledge conceptualizes entities of a complex world (Campbell, 1998, 5-9) using the complexity of the human mind as its primary tool (Waschsmuth, 1991, 4). The complex nature of knowledge will present some critical challenges to the implementation of the SWEB from the knowledge acquisition, knowledge representation and computational (reasoning/inference) perspectives.

There have been many research efforts dedicated to the understanding of complexity, and as a result, several types of complexity have been developed including crude, computational, and effective complexity (Gell-Mann, 1997, 5-19). It is not important to understand complexity theory in its entirety, as this is beyond our scope, but we should, however, be able to understand why complexity should be included in our common meaning of the concept of knowledge and what challenges and value it imposes on the widespread adoption of the SWEB.

To accomplish this we will examine Crude Complexity, the simplest type of complexity, and analyze it with respect to the above knowledge attribute set in an attempt to discover additional connections with other members of the attribute set originating from complexity. Crude complexity will be referred to simply as complexity within the context of this analysis for purposes of being succinct[5].

(1) An Analysis of Complexity. Complexity regardless of its definition is not an entirely intrinsic property of the entity being described (Gell-Mann, 1997, 5). Traits of complexity also depend on the agent describing the entity. Crude Complexity, one of many types of complexity, is defined as the length of the shortest message describing the entity **(*description*),** the level of detail at which the entity is being described **(*granularity*)** and the language employed **(*representation method*)** to communicate the description (Gell-Mann, 1997, 5). The minimum length [of the message] will also be affected by the knowledge and understanding of the world that is assumed **(*foundational or background knowledge*),** and can therefore be left out of the description by the descriptor (Gell-Mann, 1997, 5). The descriptor must assume his target user has some foundational knowledge embedded or accessible when interpreting the KR or all descriptions would be of much greater length and therefore of higher complexity. If this assumption was not made all descriptions would then be required to contain everything about the world even if it was common knowledge within the domain. Since common knowledge can be described as "I know what you know and you know that I know you know", I, the descriptor, can then make the assumption you know certain "things" and allow you to expand the description with your common knowledge vice explicitly restating what you already know in my description. In short, the property of common knowledge, which will be explained in greater detail in a later section, allows descriptors to compress their descriptions (Gell-Mann, 1997, 8).

To illustrate an example of this point a child's description of an ordinary triangle, would likely impose less complexity from its description than that of a mathematician's. As we would expect the child would rely on his experience and

[5] This type of analysis can be applied to more sophisticated and different forms of complexity, such as computational and effective complexity, to potentially discover additional associations. This however is beyond the scope of this study, as the intent of this analysis is to deliver breadth of knowledge attribute traceability without too much focus on a single one.

background knowledge about triangles to form the description. The descriptive method applied by the child would likely be aspects of common knowledge to people of like experience levels, and require little, if any additional background knowledge to ***interpret*** the description. The child would likely describe the triangle by its physical properties including color, size, number of sides, or at least in the most obvious terms. Additionally, the description would likely be shorter than the mathematician's in part due to language choice. It should be evident the mathematician's description of a triangle will be at the other end of the spectrum from the child's description and impose much more complexity by the description, language, granularity and required background knowledge to interpret and understand the description. This simple example illustrates how the description of an object, the triangle, can become more complex than its original state by the act of describing it in language. The process of describing domain concepts, as in an ontology, with KR is equally susceptible to this problem.

b. Dispersed and Disorganized

(1) Knowledge Dispersion. The dispersion characteristic of knowledge directly counters the ability of information systems to create an adequate knowledge density from which to create new knowledge. Freidrich Hayek made the observation that "…the knowledge of circumstances of which we make use never exists in concentrated or integrated form, but solely as *dispersed* bits of incomplete and frequently contradictory *(consistency)* knowledge which all the separate individuals possess (Hayek, 1945, 519). Hayek was a prominent economist speaking about the dispersed nature of the knowledge required to make economic decisions, but in general his words resonate regarding knowledge usage for application to any problem space, including the military domain (Schmitt, 1997, 232). The dispersed nature of knowledge presents a formidable challenge for any potential knowledge user and provider. Hayek suggests the restated problem to be a question of "… how to secure the best use of resources known to any members of society, for ends whose relative importance only these individuals know" (Hayek, 1945, 519) as the focal point for mitigating dispersion. Hayek implies a form of recognition or self description assisting the user with more precise searches (Davies, 2003, 3) to locate satisfactory content. This will enable the web

to aggregate large networks of machine interpretable human knowledge increasing the ability to achieve an increased content density from which to construct knowledge as the SWEB promises. In addition to Hayek's observation about self description he also implies a mechanism to "secure" or capture the resources required. The SWEB will render the knowledge user the ability to acquire the content he is seeking by providing layered interoperability amongst data and information sources and techniques for establishing common meaning.

In the Information Age the dispersed aspect of knowledge has been mitigated somewhat by the "connectedness" of the World Wide Web that effectively establishes a highly accessible knowledge pipeline and storage system (Davenport, 2000, 18). People and organizations must make the choice to commit to contributing represented knowledge to these potential vast stores in order to truly take advantage of the "network effect" and exploit an improved knowledge density. Dispersion however, will still exist in some form. To paraphrase Hayek's words, "To assume all knowledge to be given to a single mind is to assume the problem away and disregard everything that is important and significant in the real world" (Hayek, 1937, 528). We must understand no single mind, single network or single knowledge base can ever possess all knowledge, therefore the best we can hope for is to have efficient knowledge discovery and effective means to secure it. The knowledge is in the network, the SWEB must enable sufficient representation and the ability for agents to capture and apply it.

(2) Knowledge Distribution. Consequently, as the knowledge pipelines and storage systems are enabled by networks, the dispersion of knowledge becomes less of a spatial or geographic problem and more of an issue of organization and distribution. The uneven distribution of knowledge has been the subject of recent studies focused on understanding how knowledge moves about an organization and the peculiarities associated with its accumulation (Nissen, 2002, 251). Knowledge distribution within a given organization finds some components of the organization receiving a surplus of knowledge and others constantly in deficit (Davenport, 2000, 40). The uneven distribution problem is termed asymmetry by Davenport and Prusak (Davenport, 2000, 40-41) and certain amounts of it must exist for knowledge to be

valuable. Therefore, this property of asymmetry or scarcity is responsible for associating value with knowledge in an organization. The more asymmetric the knowledge, the more valuable it is as long as the required demand exists. Hayek's observations regarding the dispersed nature of knowledge suggest knowledge can be ***transferred*** to some degree. Therefore if knowledge can be transferred to organizations requiring it, asymmetry can be marginalized. The SWEB can provide the critical transfer vehicle, by applying the idea of publication and subscription to required knowledge content. So, if nearly unrestricted access to un-quantifiable amounts of web content exists and networks connect more than ever before, why does this perceived scarcity still exist? Let us examine how knowledge is distributed to determine a source of knowledge scarcity. Maybe it is not scarcity at all.

(3) Data Overload. The knowledge scarcity existing in today's World Wide Web is not caused by a shortage of raw materials to generate knowledge; the quantity of data available is at an all time high and growing, but the data is incapable of communicating meaning without appropriate an KR language. There are currently billions of web documents and databases accessible through the World Wide Web (Davies, 2003, 1). So if the raw materials to generate knowledge are plentiful how can knowledge be scarce? The perceived scarcity actually arises from the plentiful nature of data, or the data glut as it is termed. Some refer to this data glut as "information overload", but as we shall further explain, it is really data overload ([2] Pohl, 2000, 3).

Figure 6 illustrates the data overload concept by highlighting the correlation between the volume and value levels of unstructured data, structured data, information and knowledge. As volume decreases the value increases as is consistent with Davenport and Prusak's observation that the more scarce an entity the more value it has, providing the appropriate levels of demand (Davenport, 2001, 25). The roots of this problem can be traced to the data centric purposes for which computers were designed (Pohl, 2000, 5). Computers were designed to process data exclusively and were sometimes referred to as data processing centers ([2] Pohl, 2000, 5), therefore does the computer require redesigning to facilitate information or knowledge generation?

Today however, due to its recognized value, the proliferation and widespread use of the Relational Data Base and to a lesser extent the Extensible Markup Language (XML), illustrates an effort to bring a level of organization and structure to data. Data's inability to convey meaning ultimately fixes a maximum value level data can attain as is consistent with Figure 6.

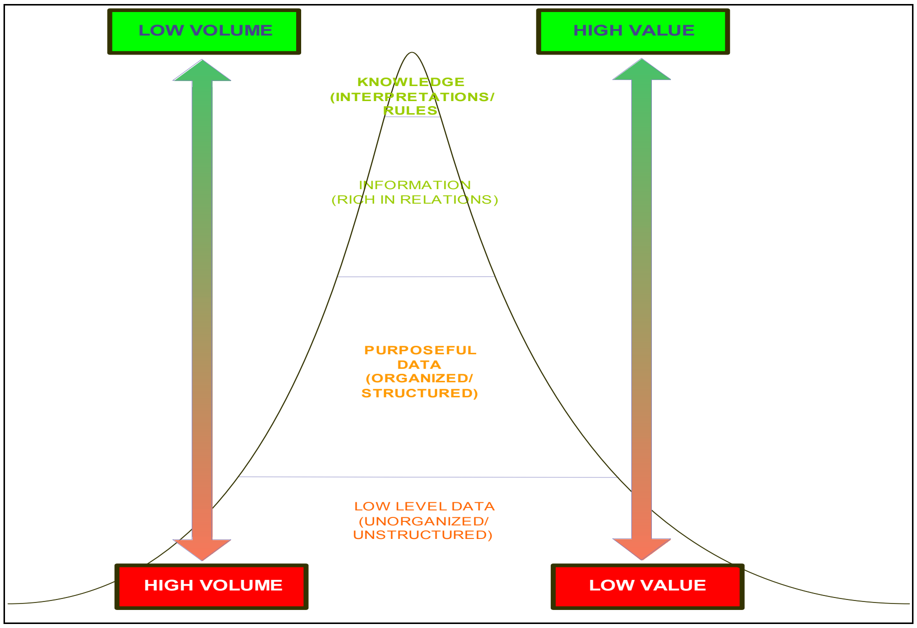


Figure 6. Information Overload Myth (From: [2] Pohl, 2000, 4).

By acknowledging the existence of, and access to the raw material, the shortage is narrowed to the conversion of these raw materials to knowledge. To understand this concept we must differentiate between data and information. Next we will introduce the traditional Knowledge Hierarchy to examine the processes required to convert data to knowledge, demonstrating why data overload is occurring. The solution to the data glut and knowledge scarcity lies in the ability to enable computers and networks to assist with rapidly and efficiently converting this data glut into knowledge, and then facilitating and managing the flow to ensure demands are met. The SWEB offers a partial solution to this problem.

(4) Data. To make the distinction between data and information we will perceive data as numbers and words without relationships (Pohl, 2000, 7). Bits and bytes stored in some raw state requiring additional processing to be of

utility to the decision maker. The processing required will be one which establishes some level of relationships between the data items. Furthermore, data can relate very little about its own meaning (Pohl, 2000, 6) and relies on mappings to a model, or series of models, with respect to the intended meaning and relationships for interpretation (Orbst, 2003, 105). These models alluded to are the ontologies, one of the core technologies in representing knowledge within the SWEB, which we have yet to discuss. As Davenport and Prusak further explain, "There is no inherent meaning in data." "Data describes only a part of what happened; it provides no judgment or interpretation and no sustainable basis for action" (Davenport, 2000, 3). Interpretation is the value added process as interpretation is where the relationships are developed. Therefore, Data + Interpretation = (Something of Value), where interpretation is a process establishing the required relationships adding more value. With proper interpretation an organization or individual can use the data as it was intended.

Data, in addition to being voluminous can also be unstructured. This absence of structure introduces a new set of problems the SWEB must contend with to build its knowledge foundation. Unstructured data is very difficult to manage and because of its unstructured nature, has little value[6] (Pohl, 2000, 6). Too much data creates the glut, and if the data is unstructured its utility can be marginalized. The marginalized utility of data makes it extremely difficult and time consuming to identify and interpret the data of value to a particular application (Davenport, 2000, 3). Structure however, can be imposed on data through data models, database management systems, schemas, the Extensible Markup Language (XML) and ontologies. These technologies will provide the required structural foundation for the SWEB, and will be elaborated on in the Data Sources Chapter. More specifically, we discuss the concept of structuring data and further develop the concept in terms of Relational and XML Databases. But according to Obrst, "Structure itself, though important is not the crucial determining or characteristic factor for the continuum: interpretation is." "Structure is just a side effect for the degree of interpretation required" (Obrst, 2003 104). Refer to Figure 7 for a

[6] We would submit data has little value in the context of automated, programmatic manipulation and interpretation. More specifically, software agents and reasoning engines would not be able to interpret.

graphic representation. To review, we have identified interpretation as the key to data utility and therefore value. Let us continue our analysis to discover the source of interpretation.

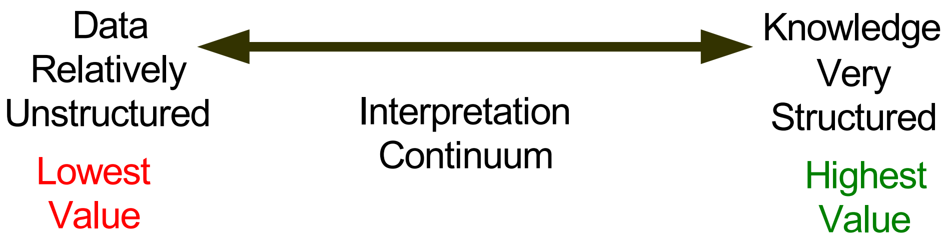


Figure 7. Interpretation Continuum (After: Obrst, 2003, 105).

(5) Information. Information is meant to change the way an entity perceives something through an impact on his behavior or judgment (Davenport, 2000, 3). Davenport and Prusak credit information with 'giving ***shape***' and directing a manner of perception to the interpreter. This "shape" levied by information combined with what the interpreter already knows, or at least has access to, creates ***actionable*** "new knowledge" (Obrst, 2003, 105). The "shape" provided by information enables interpretation. Interpretation establishes the missing relationships between data items creating utility, value and meaning. The role of interpretation becomes the mapping between a subset of data and a model of some set of objects in the domain with respect to the intended meaning of these objects and the relationships between the objects (Obrst, 2003, 105). Therefore, all of these items must combine in a process to create new knowledge. A key point embedded within this process is that in order to create "new knowledge" an organization or individual must already be in possession of some quantity of background knowledge. Without old knowledge, new knowledge cannot be generated (Obrst, 2003, 105). Additionally, to employ knowledge in any usable way an agent (human/software/machine) must possess access to the prerequisite background knowledge or intellect required to interpret the knowledge for a specific problem (Obrst, 2003, 105). This idea is captured by the refinement of the equations presented above. We now have the required background to understand what Obrst, Davenport, Prusak, and Pohl mean when they refer to knowledge.

(6) Knowledge Equation Refined. We understand Data + Interpretation = (Something of Value). We can now substitute knowledge for Something of Value, changing the equation to read Data + Interpretation = Knowledge. The equation can be manipulated to show Knowledge – Interpretation = Data. Furthermore, Interpretation = Relationships communicated by Information; therefore, the degree of interpretation is a function of the relationships established by receipt of new information combined with the knowledge we already know. Therefore, we can again restate the equation to read New Knowledge = Old Knowledge + Information[7]. With this we have navigated the knowledge generation process and understand how data, information and knowledge are related and differentiated. Interpretation emerged as the critical component of both knowledge generation and employment.

Thus far we can conclude the current knowledge generation process is unable to efficiently produce knowledge due to the inability to interpret the vast stores of data. The end result is the perceived scarcity of knowledge and a glut of data. The SWEB will focus its efforts enabling this interpretation by leveraging technologies to describe machine interpretable content through knowledge representation (See Figure 8).

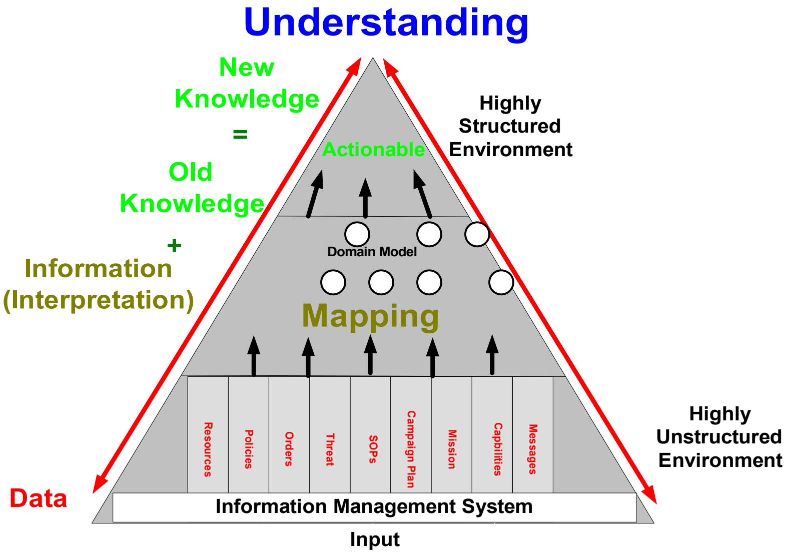


Figure 8. Knowledge Hierarchy (After: Chorfas, 2002).

[7] The knowledge equations and their various forms were reproduced from (Orbst, 2003, 105).

(7) Distribution. To correct the uneven distributions caused by the dispersed nature of knowledge, one must focus on how content flows within today's vast networks. Knowledge can be viewed in the System Dynamic perspective as either a stock (storage receptacle), or a flow (the moving content of a pipeline). To monitor the status of both stock levels and flow rates SWEB technologies can be implemented to track stock levels, look for shortages and connect with potential sources. In practice this works, but what if we wanted to allow knowledge to flow in from an external source outside our enterprise? As long as we were using the same data structures, had the same meaning for terms, and had exactly the same business process it may be effective. But, as is more commonly the case an interoperability problem will inhibit us from connecting our knowledge flows and sharing the contents of our knowledge stocks. Today there are many more technologies, tools and techniques improving this problem immensely from where it was a few years ago, but we are no where near what the SWEB promises, the promise of transparent interoperability.

(8) Knowledge Embeds. As is the case with reality, there is never a clear cut logical classification for anything (Sowa, 2000, 356). It would be simple if were classified as either a stock or flow, but knowledge embeds in routines, processes, practices and norms (Davenport, 2000, 5). Often knowledge embedded in such fluid aspects of an organization can be invisible to the organization (Housel, 2001, 9). Invisible can be equated to undiscovered knowledge, and if knowledge is undiscovered it cannot be represented. The knowledge acquisition efforts must be rigorous enough to expose invisible knowledge to an organization to fully take advantage of the value added of the SWEB. If knowledge acquisition methodologies remain undeveloped or are not followed, the SWEB could be perceived as not returning value to such organization. Knowledge must be represented in machine interpretable form for it to be leveraged. As such, due to the nature of this property of knowledge the SWEB must provide the flexibility and expressiveness in its knowledge representation languages to describe these forms of knowledge as well as provide rigorous knowledge acquisition

methodologies. The knowledge acquisition methodologies must be simple to use yet robust enough to discover invisible knowledge embedded within a process, norm, practice or routine.

(9) Knowledge is Dynamic. The dynamic nature of knowledge or its ability to change will also provide implementation challenges for the SWEB. Ontologies developed today may require a complete overhaul tomorrow. Therefore, the SWEB and its supporting technologies must be able to adapt to change by incorporating modular and extendible design patterns. These design patterns must be modular in nature and loosely coupled, with a clear separation of concerns to guarantee such. Many of the design patterns and considerations can be borrowed from Object Oriented Programming (OOP) and adapted for the knowledge handling mechanisms of the SWEB. Since the representation, storage, or acquisition of knowledge is never the end goal of a SWEB application, adopting the design patterns of OOP is a logical step since OOP applications will likely be interfacing and ultimately apply the knowledge objects the SWEB will be enabling.

C. COMMON AND DISTRIBUTED KNOWLEDGE (Φ)

Our analysis from above has left us with a long list of knowledge characteristics that through our meaning triangle have become references to the concept of knowledge. Just as there are many characteristics helping to ground the concept of knowledge within our conceptual framework there are also different knowledge types that inherit these characteristics. Two of these knowledge types the SWEB will rely on are the concepts of common and distributed knowledge. While common and distributed knowledge are subclasses of knowledge they have their own unique properties. Common and distributed knowledge manifest themselves as useful leverage points to mitigate complexity and foster an understanding of complex situations in involving members of a group or domain (Fagin, 1995, 3)

1. Common Knowledge

Common Knowledge ($C\phi$) is “what any fool knows” (Fagin, 1995, 34). More formally, it can be expressed as the mutual knowledge among a set of agents

(Vanderschraaf, 2002). When we make the assumption about the existence of some quantity of $C\phi$ within a domain, we can leverage the fact all members of the domain "know" the propositions contained within $C\phi$. Because of this those propositions can be intentionally omitted from the Knowledge Representation products subscribed to by the domain. Recalling our earlier analysis of crude complexity the result of this assumption is an elimination of the complexity caused by restating such propositions already contained in $C\phi$ in the knowledge representation. As we might infer because of these omissions the length of the descriptions are shorter. We can also claim because $C\phi$ exists within a group or domain (G), its subgroups also possess $C\phi$ (Fagin, 1995, 34). This is mathematically expressed as $C_{\mathbb{G}}\phi \rightarrow C_{\mathbb{G}'}\phi$, if G′ is a subgroup of G. This claim supports the concept of inheritance by subordinate classes

To legally declare the existence of $C\phi$ we must be able to assert all members of G know ϕ is true and is common knowledge among the group. To restate, we must be able to declare with confidence that "I know that you know, and you know that I know a certain domain proposition to be true."

Within the Military domain there are many forms of $C\phi$ we can assert with relative confidence due to our common experience, training, and operating environment. Even across the services a certain amount of $C\phi$ can be asserted, determined by our common military culture. As a result, when representing knowledge within the military domain, careful analysis should be done to identify the amount and type of $C\phi$ a user group possesses in order to maximize the reduction of complexity and still achieve a common meaning.

2. Distributed Knowledge

Distributed Knowledge ($D\phi$) can be viewed as what a "wise man would know" (Fagin, 1995, 36). More formally stated $D\phi$ is ability of agents to pool their knowledge toward some problem space (Fagin, 1995, 24). Therefore if two agents combined their knowledge they could only attain ϕ, assuming ϕ was divided and distributed among the two agents and no single agent individually knows ϕ (Fagin, 1995, 3). One might conclude if one cannot gain new knowledge from the existence of Dф why should we concern ourselves with it? As it turns out Dф allows us to deduce facts within a domain as we shall demonstrate with an example (Fagin, 1995, 3).

3. Common and Distributed Knowledge Combined

To illustrate the usefulness of Cф and Dф within the military domain consider an array of sensors deployed from the Remotely Monitored Battlefield Sensor System (REMBASS). The REMBSASS system is a tactical system consisting of seismic (Ss), acoustic (Sa), and magnetic (Sm) sensors. The sensors are designed only to determine the presence of an object based on its respective detection capabilities. In addition to presence, the sensor can also classify the detection in a low, medium or high category based on the strength of the detection compared to predetermined baseline thresholds.

Suppose an independent intelligence report is received by a group of analysts responsible for interpreting our sensor array's data. The report states there are T-72 Tanks and infantry units in the Area of Responsibility (AOR). All of the analysts believe this report, as it originated from a credible source. That assertion about the presence of tanks and infantry units within the AOR can now becomes Common Knowledge among the group of analysts as long as all members of the group know that all members of the group believes and knows this information. Suppose Ss detects a ground vibration classified as high, and Sa detects an acoustic event classified as high, and Sm detects the presence of metal classified as high. These facts taken collectively and combined with the Common Knowledge of the group can lead to the conclusion the sensors have detected at least one T-72 tank. The sensor data taken individually would be incapable of forming this conclusion, as they would only be able to assert the facts within their detection capabilities. The combination of Common and Distributed Knowledge is where we acquire leverage.

Additionally, because of the high classification of the metal and seismic detections the analysts are able to exclude the presence of infantry units only. The analysts cannot however conclude that there is not Infantry Units accompanying the tanks, as the detection of the tanks overpowers the signature of the infantry units. Notice how Common Knowledge was combined with Distributed Knowledge leading to some very useful conclusions that otherwise would not have been possible.

D. KNOWLEDGE MANAGEMENT LIFECYCLE

Thus far we have formed a common meaning of the role knowledge will play and what it means in the context of the SWEB. From our analysis it is clear a common process must be established to manage the knowledge lifecycle (See Figure 9). Many different knowledge management lifecycles exist most of which are strikingly similar. The Knowledge Management Lifecycles synthesized by Nissen from a variety of sources (Nissen, 2002, 255) all seem to generally agree in concept and differ only in terminology. To emphasize the importance of a standard management methodology we will analyze our proposed methodology in detail by phase throughout our work. By addressing the necessity of a Knowledge Management Lifecycle in the knowledge section we hope to expose the reader to this very important concept to heighten awareness as we present more detailed analysis.

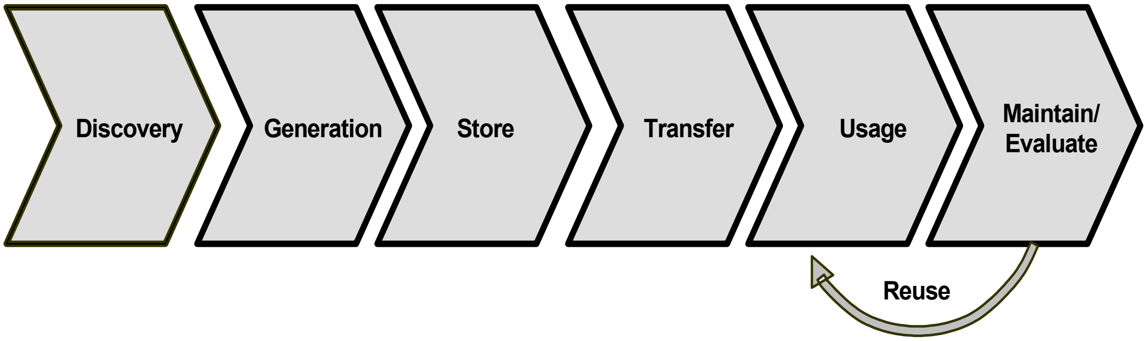


Figure 9. Process for Managing Knowledge Lifecycle (After: Nissen, 2002, 255).

E. SUMMARY

Military users will soon be able to take advantage of the SWEB as it endeavors to harness the power of knowledge by introducing techniques to construct knowledge from enabled network content and represent it in a manner interpretable by computers. In an attempt to summarize the importance of the promise of being able to construct knowledge from available network content, the foundation of the SWEB, we will examine some of the interactions of the characteristics of knowledge we have previously analyzed to show when taken as a system other behaviors emerge. The Causal Loop Diagram (CLD) (Figure 10) demonstrates these facts and is a useful analytical tool established by the Systems Dynamics community. The below section is written to be used in conjunction with the CLD depicted below. We recommend the reader follow along with the diagram as the description is read to receive the full benefit of this summary

1. Causal Loop Diagram

The knowledge construction process enabled by the SWEB will create new knowledge from existing foundational knowledge by combining it with the appropriate interpretation. The new knowledge will then be transferred to the intended users, but by its nature will be dispersed creating shortages and surpluses varying the amount of knowledge able to be constructed or knowledge density. The improper distributions and resultant shortages and surpluses will drive up demand for knowledge, thus increasing its value creating a positive feedback loop. The natural dispersion of knowledge will allow only modest gains to be achieved in knowledge density and knowledge yield. The result will still be and increased net decision rate, but only of modest gains. The true value added of the SWEB will be the construction of knowledge by aggregating enabled network content, mitigating dispersion and ensuring the constructed knowledge is properly distributed resulting in the right knowledge to the right place at the right time. If this is accomplished knowledge density will remain high, as well as knowledge yield, allowing greater increases to be realized in the net decision rate. To accomplish these daunting tasks users must have a thorough understanding of the complexities of constructing knowledge and its characteristics, as this is the foundation on which the SWEB will be built. This will be a critical enabler toward helping to mitigate the uncertainty of warfare and allowing the military to achieve the goal of becoming a fighting force based on knowledge.

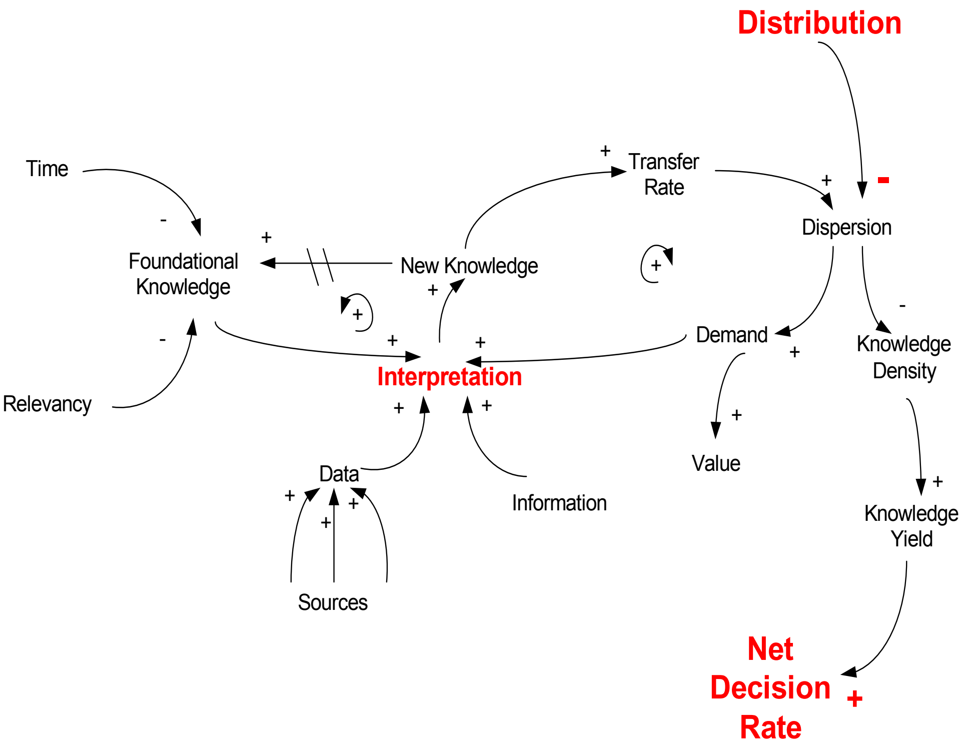


Figure 10. SWEB Causal Loop Diagram.

THIS PAGE INTENTIONALLY LEFT BLANK

III. NETWORK DATA SOURCES: BUILDING BLOCKS OF KNOWLEDGE

A. DATA SOURCES: THE BUILDING BLOCKS OF KNOWLEDGE

This chapter serves to underscore the importance of data sources residing on a network to the Semantic Web's (SWEB) knowledge generation process. We discuss various types of data sources likely encountered and methods to successfully enable them. The more we understand about a data source the more informed decision we can make as to whether or not to put forth the effort to incorporate it into our application. Key points to take from this chapter include the implications of structured data, and how the Extensible Markup Language (XML)[8] relates to different database models. Notably, data may be described as the atomic element of knowledge; how we structure and store data is foundational to creating semantically represented, machine-readable information.

The proliferation of the World Wide Web (WWW) has lessened the requirement for systems and users to store, maintain and own the data driving their information and knowledge systems. Today the WWW affords us the luxury of connecting to the expert's data sources to support our applications. In the military we may connect to a weather data source owned and maintained by weather experts and Order of Battle (OOB) data sources owned and maintained by the intelligence agencies. This data is directly from the experts and adds tremendous value to the effectiveness and efficiency of our data driven applications supporting military operations. One down side is that there is more data populating the WWW than can presently be efficiently and effectively used.

Another drawback, or limitation, is the fact that the network is not (completely) reliable.[9] Consequently, our generic SWEB architecture prescribes a local mirror, or cache, of the network data sources comprising our application. The employment of a caching mechanism serves as a temporary buffer mitigating adverse effects resulting from network disruptions. For example, loss of a networked data source could preclude processing some function within our application. Storing the incoming data in a local

[8] This paper assumes a basic working knowledge of XML. Many sources are available to learn XML including [http://www.w3c.org/xml].

[9] In fact, as we observe in the Distributed Computing chapter, the idea that the network is reliable leads to a fundamental design error in traditional networking applications.

cache allows us to continue processing for a finite period before the data value perishes. The mitigating effect of the locally caching data sources will vary proportionately with the time-value of the data. Although not a perfect solution a local mirror of our data sources does benefit the overall application.

As we will see in later chapters, the ability to leverage network data sources is critical to the wide spread adoption of the SWEB. Unfortunately, the extent to which we can interact with another agency's data source is a function of what we know about the data source. To more effectively and seamlessly use a network data source machines or software agents must know its structure, content, format, and how the data should be interpreted or its meaning. Without this knowledge the probability of effectively incorporating the remote data source into our own program is low. So what mechanisms can we implement to allow potential data users to fully exploit the data source(s)? The answer resides in common meaning, knowledge representation, and XML; all of which are part of the SWEB.

To incorporate a network data source we must first discover it. After discovery we must put forth the effort to enable it to meet our needs. To do this we must analyze the data's requisite structure, or lack thereof. Data can be classified into three general forms: structured, semi-structured and unstructured. Fortunately, we can impose structure at several different sublevels in the data source depending on the shortcomings of the data. The leverage points we can activate include the schema, data model, data type or on the literals themselves. We will now elaborate on each of these data classifications and further discuss the potential leverage points.

1. Unstructured Data Source

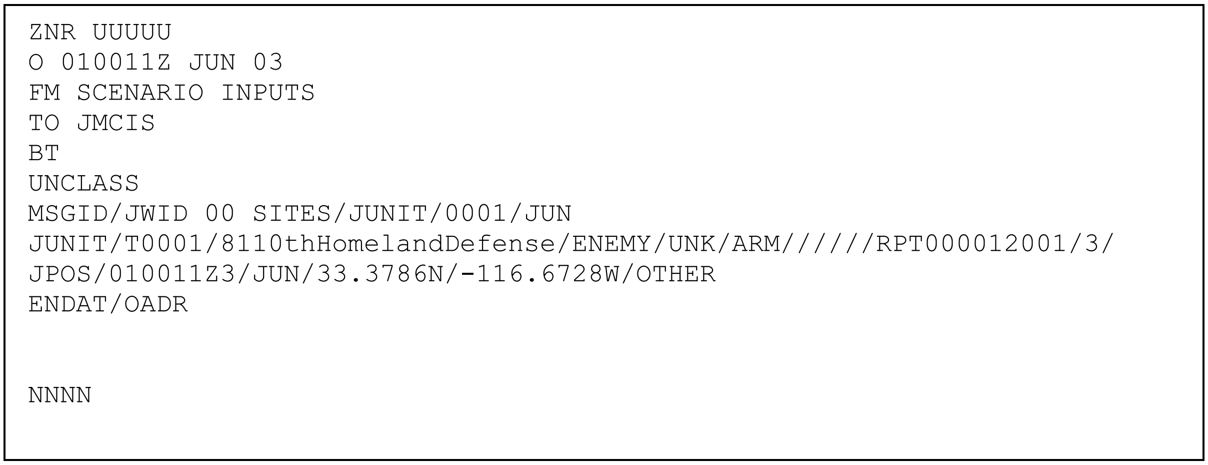
Unstructured Data can take considerable effort to transform into a workable format and even with a massive effort there is no guarantee the source may be adapted. Unstructured data assumes no recognizable or intelligible form to the interpreter. In some cases the data source may in fact be structured, but the interpretation instructions are missing, in which case the data is of no use. Recently, the likelihood for encountering purely unstructured data on the WWW has drastically decreased as enterprises have come to recognize the value of their data, and have applied great efforts to make their data usable by bringing structure to it. Therefore, missing interpretation instructions are the most probable cause for data to be classified as unstructured.

An example of unstructured data may be a memo or letter. While an implicit structure may be inferred by a human, machine processing and interpretation is essentially limited to string matching. This situation requires the human to remain in a critical position within the process.

2. Semi-Structured Data

Data in a semi-structured form can in some aspects be just as challenging as unstructured, but generally provides enough structure to enable at least a partial transformation. Therefore, if the data source lacks the necessary structure, we must have the ability to impose the structure we require for application to make the source work for us.

An example of a semi-structured data source might include an Over-the-Horizon (OTH) report message, as illustrated in Figure 11. A specification may define a format and structure for this message. However, the resulting text document is still only partially structured. Accordingly, although machine processing is more plausible, it is limited as rigorous semantics are not thoroughly applied. Specifically, the use of slashes to separate data elements does nothing to provide semantics. As well, machine processing is made more difficult because slashes may in fact be data elements.



```
ZNR UUUUU
O 010011Z JUN 03
FM SCENARIO INPUTS
TO JMCIS
BT
UNCLASS
MSGID/JWID 00 SITES/JUNIT/0001/JUN
JUNIT/T0001/8110thHomelandDefense/ENEMY/UNK/ARM//////RPT000012001/3/
JPOS/010011Z3/JUN/33.3786N/-116.6728W/OTHER
ENDAT/OADR


NNNN
```

Figure 11. Semi-Structured OTH Message.

3. Structured Data Source

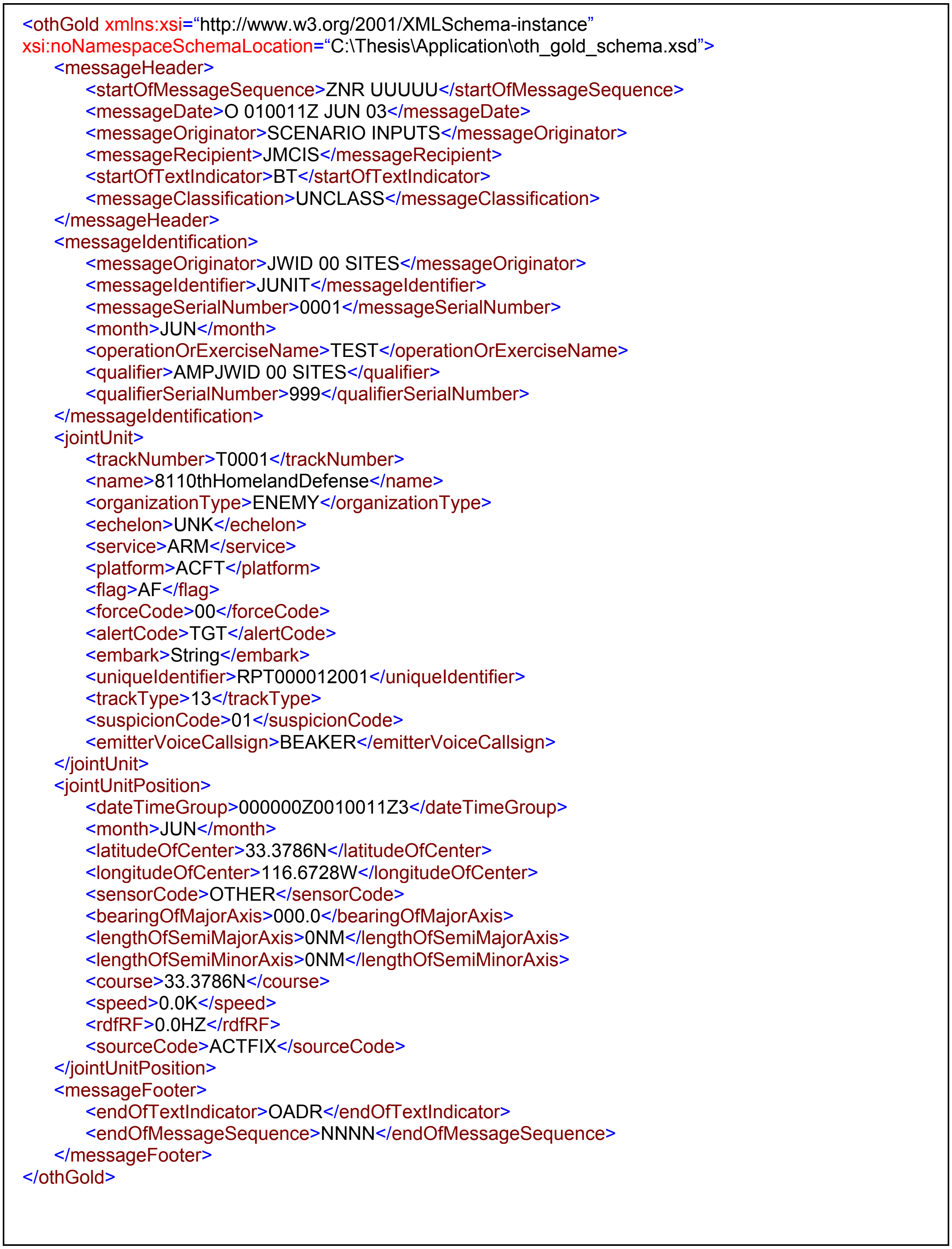
In general, the more structured the data source, the more susceptible it will be to manipulations to support a knowledge generation process. A data source classified as structured implies knowledge of the underlying data model or schema. The presence of such provides a metadata source, or data about data, allowing insight into the storage structure, cardinality and data types contained within. A formal schema does not exclusively classify the data source as structured, but without one structure is more difficult to determine.

An alternative to the presence of a formal schema is the implicit structure of the data elements themselves. Often times the data elements can be stored in a recurring, easily recognizable format such as a specific format of a message presenting an informal or unwritten schema by which to determine structure. The presence of such structure embedded in the data elements also classifies a data source as structured.

As we will see, even within structured data we can achieve varying “degrees of structure.” For example, a basic XML document essentially provides only terms and their associated values, along with hierarchy; whereas, derivative technologies[10] additionally provide classification and logic. Specifically, the Ontology Web Language abstracts description logic and provides classifications of relevant concepts in a domain. This technology will be expounded upon somewhat in this chapter, and in detail in the Ontology chapter.

The OTH message we mentioned earlier as an example of a semi-structured data source may be formatted as a well-formed XML document as seen in Figure 12. The XML document elements indicate applicable terms (semantics) relating to an OTH message and the element contents. A casual review of the XML formatted message reveals s a much greater understanding of the OTH message. Modeling our data source in a highly structured, semantic manner will significantly facilitate automated processing of the contained data.

[10] By “derivative technologies” we mean technologies that are XML at there basic level. That is, they adhere to the basic XML specification.



```
<othGold xmlns:xsi=“http://www.w3.org/2001/XMLSchema-instance”
xsi:noNamespaceSchemaLocation=“C:\Thesis\Application\oth_gold_schema.xsd”>
    <messageHeader>
        <startOfMessageSequence>ZNR UUUUU</startOfMessageSequence>
        <messageDate>O 010011Z JUN 03</messageDate>
        <messageOriginator>SCENARIO INPUTS</messageOriginator>
        <messageRecipient>JMCIS</messageRecipient>
        <startOfTextIndicator>BT</startOfTextIndicator>
        <messageClassification>UNCLASS</messageClassification>
    </messageHeader>
    <messageIdentification>
        <messageOriginator>JWID 00 SITES</messageOriginator>
        <messageIdentifier>JUNIT</messageIdentifier>
        <messageSerialNumber>0001</messageSerialNumber>
        <month>JUN</month>
        <operationOrExerciseName>TEST</operationOrExerciseName>
        <qualifier>AMPJWID 00 SITES</qualifier>
        <qualifierSerialNumber>999</qualifierSerialNumber>
    </messageIdentification>
    <jointUnit>
        <trackNumber>T0001</trackNumber>
        <name>8110thHomelandDefense</name>
        <organizationType>ENEMY</organizationType>
        <echelon>UNK</echelon>
        <service>ARM</service>
        <platform>ACFT</platform>
        <flag>AF</flag>
        <forceCode>00</forceCode>
        <alertCode>TGT</alertCode>
        <embark>String</embark>
        <uniqueIdentifier>RPT000012001</uniqueIdentifier>
        <trackType>13</trackType>
        <suspicionCode>01</suspicionCode>
        <emitterVoiceCallsign>BEAKER</emitterVoiceCallsign>
    </jointUnit>
    <jointUnitPosition>
        <dateTimeGroup>000000Z0010011Z3</dateTimeGroup>
        <month>JUN</month>
        <latitudeOfCenter>33.3786N</latitudeOfCenter>
        <longitudeOfCenter>116.6728W</longitudeOfCenter>
        <sensorCode>OTHER</sensorCode>
        <bearingOfMajorAxis>000.0</bearingOfMajorAxis>
        <lengthOfSemiMajorAxis>0NM</lengthOfSemiMajorAxis>
        <lengthOfSemiMinorAxis>0NM</lengthOfSemiMinorAxis>
        <course>33.3786N</course>
        <speed>0.0K</speed>
        <rdfRF>0.0HZ</rdfRF>
        <sourceCode>ACTFIX</sourceCode>
    </jointUnitPosition>
    <messageFooter>
        <endOfTextIndicator>OADR</endOfTextIndicator>
        <endOfMessageSequence>NNNN</endOfMessageSequence>
    </messageFooter>
</othGold>
```

Figure 12. XML Formatted OTH Message.

Structuring data has long been fundamental to automation and computing. Data based computing has implemented innumerable models and techniques for describing and storing data. However, throughout the progression of data based computing there have been at least four constants concerning data:

- Data must be stored somewhere, and it must be retrievable
- Data must be presented in some useful manner
- Data must be accessed, manipulated and transformed from its storage state and location to its consumption state and location.
- The previous must not only work well at each stage, but also remain flexible.

These constants say nothing about the actual data or any possible internal relationships; it is assumed they will be captured. Instead they refer more to the environment the data exists in and operates to form a viable solution to a problem. (Williams, 2002, 10)

Indeed, it is the need to acquire knowledge to solve problems that serves as the impetus for seeking data and its subsequent meaning. The premise that structured data is necessary to the construction of knowledge leads us to the need to select a data-structuring mechanism. XML provides a well-suited means to structure data, and moreover, to help represent knowledge. Although other mechanisms are possible, XML is by far the most commonly implemented, open-standard language for structuring data. This is at least partly because XML may be readily implemented to support a host of different applications. Additionally, extensibility is the core attribute of XML, and it presents a highly flexible structure. By flexible, we mean that it may be strictly structured as required for a particular application, yet completely transformable to satisfy other uses. Because of the importance of data in computing applications and the utility of XML for structuring data, this chapter discusses aspects of enabling data sources using XML as an interim step prior to formal Knowledge Representation.[11] It describes approaches to data storage and manipulation using relational models, XML, and a hybrid of both. Additionally, we highlight the concept of native XML databases as an option for

[11] It is important to understand that "plain" XML will likely not be the method used to represent knowledge; instead, some derivative of XML will be implemented, such as Ontology Web Language (OWL) – which will be discussed in this paper.

semantically-enabled data sources. We understand SWEB applications will employ a combination of both relational and XML data sources to support reasoning through an ontology. It is important the reader keeps this chapter in the context of how data sources are used to support the knowledge based applications of the SWEB.

While XML is a solution for data structure and interoperability, relational databases have enjoyed a very successful history that will likely continue into the future. Presently they remain the predominant method for data based solutions such as:

- Support for thousands of concurrent users
- Store terabytes of data; of any data type (including XML)
- Support relational and analytical data models
- Provide for ad hoc, flexible queries of the data
- Feature authentication and authorization security mechanisms
- Allow for programming within the database

In addition to these characteristics, relational databases also provide network connectivity to applications and other databases, and tools to develop, manage, and report on enormous amounts of data.

Since relational databases outperform other solutions on the four constants described earlier, there is no compelling reason to discontinue implementing them (Williams, 2002, 11). In fact, SWEB applications will implement relational data bases to support, and even function as part of the Knowledge Base. Relational databases and Native XML Databases will combine to form a Hybrid Data/Knowledge Model to drive the applications of the SWEB. See Figure 13.

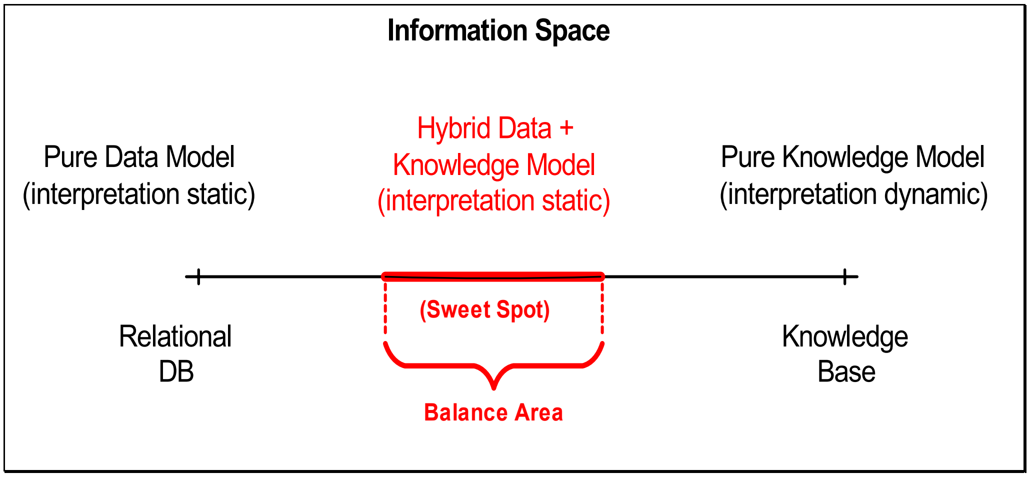


Figure 13. Hybrid Data/Knowledge Model (After: Smart, 2003).

In practice, data storage remains critical to computer applications. Further, data-driven web applications are essentially the current focus of development. SWEB applications are still in the research and academic stage; however, if the level of effort is an indicator, deployments of fully functional SWEB applications will be realized very soon.

The web has vastly expanded the amount of data shared by users. Protocols such as HTTP have provided simple and ubiquitous transport platforms to interconnect applications. Data driven sites have made the commercial web. Typical sites feature data-enabled applications allowing manipulation and presentation of data stored in databases. The browser-centric web has enabled application-agnostic interoperability wherein the user is largely relieved from the burden of maintaining compatibility with the server application(s). Processing of data for HTML presentation may be performed within the database or inside an application server resident on a web server, and propagated between client and server over (ubiquitous) HTTP. The loosely coupled, highly interoperable architecture essentially extends the web audience to global proportions and is largely software independent. Despite this great success, HTML is not optimal. Most content produced or residing on the web of today is intended for human consumption. Fortunately, HTML data may be classified as semi-structured and may be enabled for inclusion in a knowledge generation process. That said, most of the content on computers and in databases today essentially lay dormant, awaiting use. Imposing the structure provided by XML, the audience of applications can be vastly expanded for consumption by humans and machines.

XML is the foundation for the common language of conversation across the web and across platforms connected to the web. As we will see, XML representation and storage will substantially enhance interoperability, allowing loose coupling of disparate systems, such as Service and Combined systems. Semantic representation of data using the Web Ontology Language (OWL) built in XML's foundation empowers computers and software agents to interpret resident semantically enabled data. OWL will provide the format, structure and logic for sharing and reusing data across the SWEB and will result in self-describing data for software agents to use in myriad applications.[12] Increased implementation of OWL, XML and derivative technologies will necessitate new storage requirements and opportunities for innovative storage techniques. This leads to an intersection between burgeoning XML centric storage and applications and the proven relational databases.

B. XML AND DATABASES

Despite the proliferation of advanced technologies, legacy systems will continue to exist[13]. Additionally, the need to connect and communicate with extant systems will remain. The utility of XML as a key interoperability enabler is realized through its ability to allow communication between disparate systems. In addition to communicating between otherwise incompatible systems, legacy systems retain the requirement to operate within the context of their native models (e.g., relational databases). As we discuss further in the Knowledge Base Chapter, the applications of the SWEB will not be required to own the data sources driving them. The sources will exist in their native models and permit outputs exposed in structured formats such as XML, allowing simple migration to a host of other formats. A typical enterprise computing system would likely follow this pattern. Enterprise computing systems consists of multiple applications connected to one or more data sources. Not all of these applications will readily support

[12] The Ontologies chapter discusses OWL in detail.

[13] XML and Java, as we will see later, may obviate the notion of "legacy" systems.

XML, nor would they necessarily need to communicate in XML. As we stated above the data source or its output may simply be exposed as XML while retaining its native model (See Figure 14). Once the data is represented as structured XML, it may be readily communicated and shared across heterogeneous networks, platforms, and programming languages.

Implementing a database as a primary storage repository for XML documents ensures legacy applications can coexist with new XML-based applications. In fact, making data available in both XML and traditional forms essentially eliminates interoperability issues. The underlying model, or structure of the data may remain unaltered, and adding an XML view of the data, which may be modeled similarly or completely different, will have no effect on the on the relational [legacy] construct. Additionally, the data structure may change and will not necessitate a change in the exposed XML structure. Exposing relational data as XML can also serve to ease migration to XML. (Ahmed, 2001, 696-9)

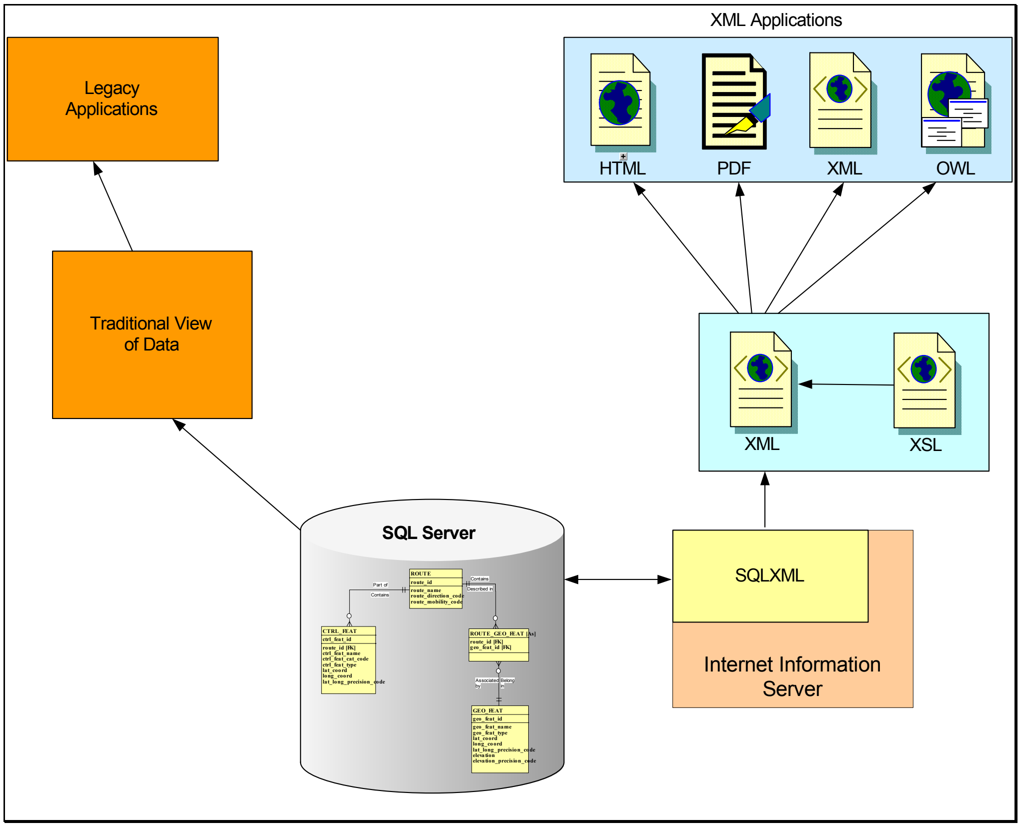


Figure 14. Relational Database Exposed as XML.

XML will never replace [relational] databases, but the two will become more closely integrated with time. (Hunter, 2001, 492) However, storage and manipulation of XML will be a necessary requirement for future database technologies. While proprietary storage formats may endure for the sake of performance, many data exchanges between applications and systems will use XML.[14] (Birbeck, 2001, 39) To be sure, many applications already use XML for data exchange, and the numbers of XML implementers continue to grow. As for existing (relational) database technologies, performance, maturity and flexibility is inherently a prime motivation for retaining this technology. Using a relational database for storing XML documents allows users to benefit from these investments. XML, on the other hand, offers the advantage of portability and flexibility.

There are alternative and stop-gap methods to introduce XML into relational databases. One method is to store XML documents as single binary or character objects without decomposing the document. The benefits of this method include document level control, and stability. However, querying below the document level is not possible. Accordingly, this storage method would necessitate additional processing to enable the storage of KR formatted data. This is a situation wherein the data is structured, yet not actually enabled.

The other method is to deconstruct the XML document into a form the database can understand. This allows for the best use of the XML document. The data within the document, persisted in a relational model, is more readily accessible by agents who can execute a variety of query languages on the data (e.g., SQL, Xpath). To accomplish this method, a mapping between XML elements and attributes to table columns must be developed. This provides the database access to the document elements and attributes, along with their corresponding contents. This approach is particularly suitable to data-centric XML documents where the structure is highly regular. This technique also has the desirable effect of closely positioning XML to the relational database. Why is this

[14] XML will likely work its way deeper into application development. Instead of only using XML to interoperate between systems, applications themselves will be developed using XML technologies to further enhance interoperability.

desirable? The answer lies in speed and agility. The "sooner" the data is structured as XML or one of its derivatives, the "sooner" it is completely interoperable with the KR of the knowledge generation process. The sooner data is semantically represented the sooner it is available for software interpretation.

The notion of legacy systems is further reduced as a design consideration or concern by interoperability at the data layer essentially obviating the need for elaborate interfaces between dissimilar systems. The implementation of an XML derivative technology called Simple Object Access Protocol (SOAP) has facilitated interoperability between different programming languages, such as Java and C++. By modeling data in a structured, textual format such as XML, dissimilar programming languages can readily parse and process the exchange the data. An instantiation of this is seen in the application area of Web Services.

Web services are precisely about interoperability and machine interpretation of KR enabled data. Web services implementing XML derivative technologies such as SOAP and Web Services Description Language (WSDL) produce loosely-coupled, highly-interoperable frameworks. In essence, web services represent distributed application computing across heterogeneous platforms.[15]

In addition to interoperability, exposing relational data as XML supports (web) application development by supporting separation of content and business logic from presentation. A common paradigm describing this notion is the Model-View-Controller (MVC) paradigm. The "Model" represents the structure of the data; the "View" is the presentation of the data; and the "Controller" is the business logic, and the logic that controls interaction between the model and the presentation. XML documents are readily transformed to various presentational formats by use of Extensible Stylesheet Language (XSL). For example, relational data structured as XML can be transformed by XSL into HTML or a number of other formats such as PDF, Scalable Vector Graphic (SVG), Extensible Three-dimensional Document (X3D) for rendering in a browser; or to OWL, which is of primary concern in the knowledge generation process and machine

[15] Various technologies such as CORBA, DCOM, RMI, and Jini enabled distributed computing; however, web services based on SOAP provide platform-agnostic, open-standards based distributed computing solutions. Distributed computing will be further discussed in Chapter III.

interpretability. Next, we discuss an XML-driven application that exposes relational data as XML and supports the notion of "separation of concerns." It is called the "Route" application as it contains data that may be used to describe a path from an origin to a destination. Briefly, it is an example of mapping a relational database model to XML, and rendering the produced XML in a variety of presentation formats, including HTML, PDF, and "plain" XML.

1. XML Driven Route Application

The Route application is loosely associated to a standardized NATO developed data model. It is called the Route Application as it is intended to describe the basic components making a military route. The routes in this application may include many control features, such as route entry points, way points, and exit points, etc. Numerous other control features are available and are explicitly defined and limited by the application schema. The application schema in this case consists of both the XML Schema[16] that constrains XML content and the Structured Query Language (SQL) Data Definition Language (DDL) schema that constrains the relational data. Additionally, a route may include many geographic features such as lakes, beaches, and mountains; geographic features are similarly limited by the schemas. As indicated by the entity-relation diagram in Figure 15, geographic features may belong to more than one route; whereas, a given control feature belongs to only one route. The significance of this model is the deconstruction/construction of XML to/from a relational database. It is not necessarily intended to precisely depict an operational model of a route. It is, however, a concrete instance of mapping XML to and from a (legacy) relational database.

[16] Our experience with XML development, even though fairly limited, has re-enforced time and again the importance of developing the data schema(s) early in the application development lifecycle.

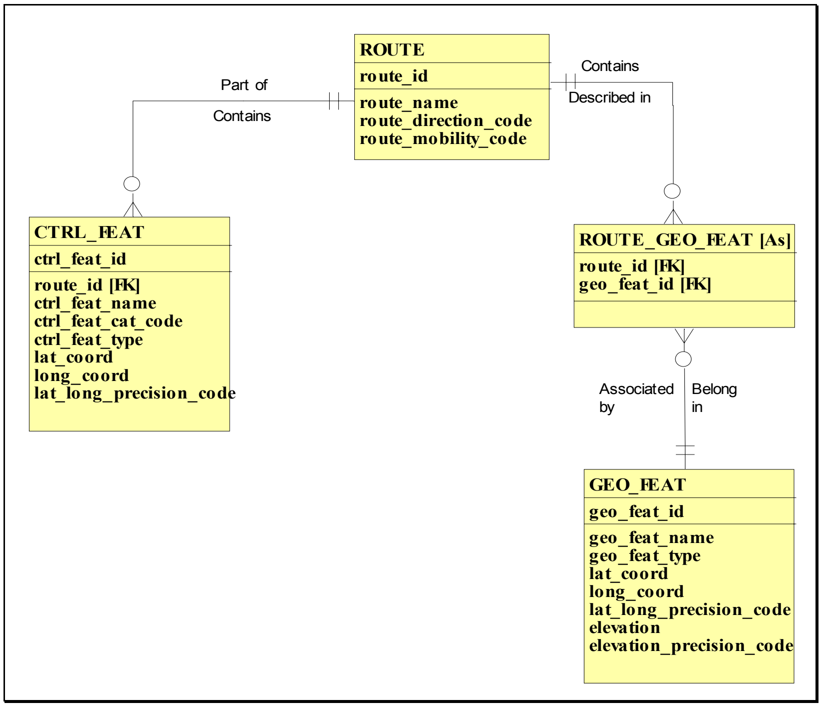
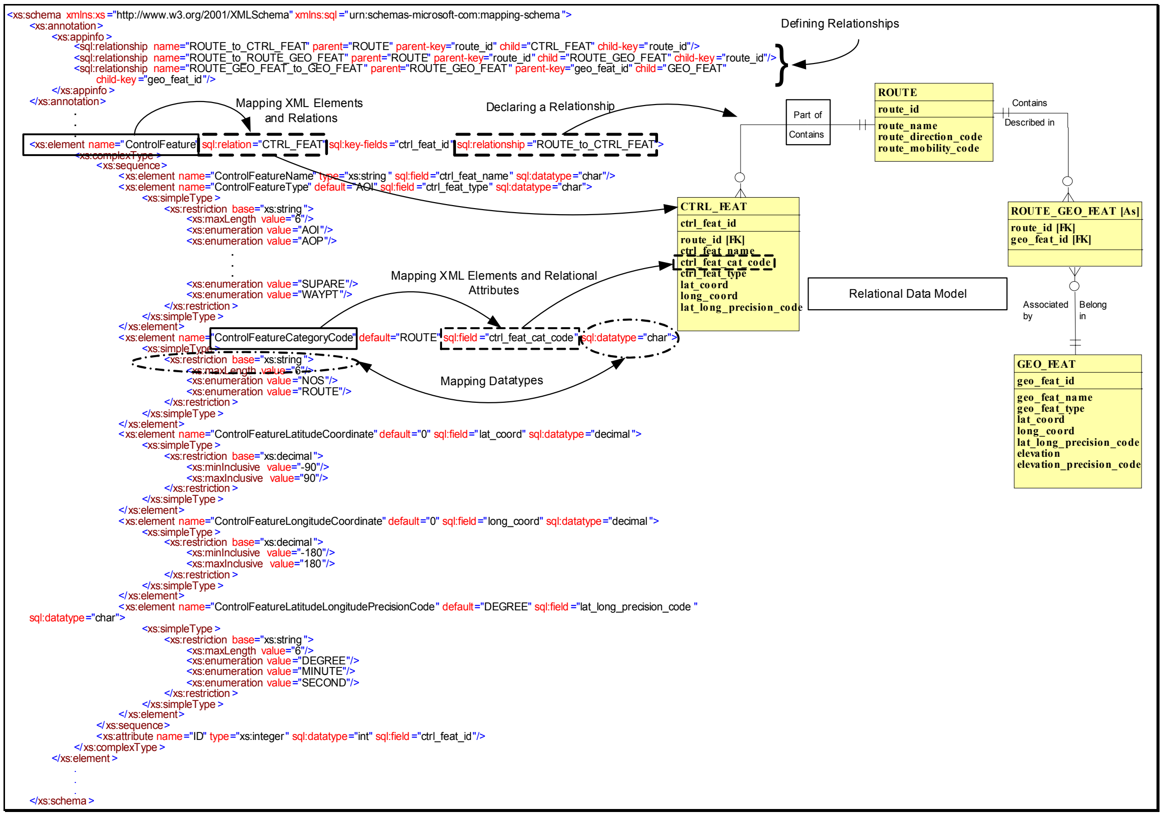


Figure 15. Route Entity-Relation Diagram.

The implementation of the (relational) data model is a SQL Server 2000 relational database. The data model consists of four entities, or tables. Each table includes attributes, or table columns that describe the entity. Specifically, the entities include a "ROUTE" table that contains overall route identification attributes, a CTRL_FEAT table that contains relevant control features for defined routes, a GEO_FEAT table that houses defined geographic features and an associative table that associates various routes with various geographic features, and vice versa.

Using the features of the SQLXML Application Programming Interface (API), we constructed an annotated XML Schema that maps the relational data to an XML instance document. Figure 16 illustrates a snippet of the annotated schema and the entity-relation diagram of the route data model. The boxed attributes in the schema represent the CTRL_FEAT table in the relational database and the one-to-many relationship between the ROUTE table and the CTRL_FEAT table. Careful examination of the mapping schema indicates the mapped XML elements and corresponding relational entity attributes. All relationships are defined in the <xs:annotation> section of the schema document. Once the relational data is mapped to the desired XML structure, it may be transformed and manipulated by virtually any application in any language, on any platform. As well, the XML structure may be marked up into semantic representation, allowing intelligent agents to understand its meaning as well as content. Of course, XML data may be rendered in traditional formats also. For example, the XML data may be transformed into HTML and rendered in a client browser, or it may be transformed into something like PDF, a more formal presentation media. Both of these renderings, and potentially many other formats (e.g. text, SVG, WAP, etc.) are dynamically produced based on the contents of the (relational) database. The Route application implements features of SQLXML and XSLT to create dynamic HTML outputs and Cocoon 2[17] to produce dynamic Adobe® PDF outputs. In a SWEB application, the exposed XML data could be transformed into an OWL instance document and interpreted by an intelligent agent employing the associated ontology.



```
<xs:schema  xmlns:xs ="http://www.w3.org/2001/XMLSchema" xmlns:sql ="urn:schemas-microsoft-com:mapping-schema ">
     <xs:annotation>
          <xs:appinfo >
               <sql:relationship  name="ROUTE_to_CTRL_FEAT" parent="ROUTE" parent-key="route_id" child="CTRL_FEAT" child-key="route_id"/>
               <sql:relationship  name="ROUTE_to_ROUTE_GEO_FEAT" parent="ROUTE" parent-key="route_id" child ="ROUTE_GEO_FEAT" child-key ="route_id"/>
               <sql:relationship  name="ROUTE_GEO_FEAT_to_GEO_FEAT" parent="ROUTE_GEO_FEAT" parent-key="geo_feat_id" child="GEO_FEAT"
                    child-key ="geo_feat_id"/>
          </xs:appinfo >
     </xs:annotation>
               .
               .
               .
     <xs:element name="ControlFeature" sql:relation="CTRL_FEAT" sql:key-fields ="ctrl_feat_id" sql:relationship ="ROUTE_to_CTRL_FEAT">
               <xs:complexType >
                    <xs:sequence>
                         <xs:element name="ControlFeatureName" type="xs:string " sql:field ="ctrl_feat_name" sql:datatype ="char"/>
                         <xs:element name="ControlFeatureType" default="AOI" sql:field ="ctrl_feat_type" sql:datatype="char">
                              <xs:simpleType >
                                   <xs:restriction  base="xs:string ">
                                        <xs:maxLength  value="6"/>
                                        <xs:enumeration value="AOI"/>
                                        <xs:enumeration value="AOP"/>
                                                  .
                                                  .
                                                  .
                                        <xs:enumeration value="SUPARE"/>
                                        <xs:enumeration value="WAYPT"/>
                                   </xs:restriction >
                              </xs:simpleType >
                         </xs:element>
                         <xs:element name="ControlFeatureCategoryCode" default="ROUTE" sql:field ="ctrl_feat_cat_code" sql:datatype ="char">
                              <xs:simpleType>
                                   <xs:restriction  base="xs:string ">
                                        <xs:maxLength  value="6"/>
                                        <xs:enumeration value="NOS"/>
                                        <xs:enumeration value="ROUTE"/>
                                   </xs:restriction >
                              </xs:simpleType >
                         </xs:element>
                         <xs:element name="ControlFeatureLatitudeCoordinate" default="0" sql:field="lat_coord" sql:datatype ="decimal ">
                              <xs:simpleType >
                                   <xs:restriction  base="xs:decimal ">
                                        <xs:minInclusive  value="-90"/>
                                        <xs:maxInclusive  value="90"/>
                                   </xs:restriction >
                              </xs:simpleType >
                         </xs:element>
                         <xs:element name="ControlFeatureLongitudeCoordinate" default="0" sql:field="long_coord" sql:datatype ="decimal ">
                              <xs:simpleType >
                                   <xs:restriction  base="xs:decimal ">
                                        <xs:minInclusive  value="-180"/>
                                        <xs:maxInclusive  value="180"/>
                                   </xs:restriction >
                              </xs:simpleType >
                         </xs:element>
                         <xs:element name="ControlFeatureLatitudeLongitudePrecisionCode" default="DEGREE" sql:field ="lat_long_precision_code "
     sql:datatype ="char">
                              <xs:simpleType >
                                   <xs:restriction  base="xs:string ">
                                        <xs:maxLength  value="6"/>
                                        <xs:enumeration value="DEGREE"/>
                                        <xs:enumeration value="MINUTE"/>
                                        <xs:enumeration value="SECOND"/>
                                   </xs:restriction >
                              </xs:simpleType >
                         </xs:element>
                    </xs:sequence>
                    <xs:attribute name="ID" type="xs:integer" sql:datatype="int" sql:field ="ctrl_feat_id"/>
               </xs:complexType >
          </xs:element >
               .
               .
               .
     </xs:schema >
```

Figure 16. Sample from Annotated Schema and Entity-Relation Diagram.

[17] See [http://xml.apache.org/cocoon] for information about Cocoon.

Another key concept to enumerate is the notion of incompleteness and lack of foreknowledge. The Route application, although it may be transformed into a semantic instance for one application/agent, it may only comprise a partial input (data) for another application. Similarly, after being semantically-enabled, the data may be discovered by an agent a priori and used by a completely different independent application.

As previously mentioned, one of the possible transformations is into OWL. This enables the relational data to be described in a KR format making it machine interpretable. KR will enable software agents to interpret contents at run time and allow the data to be described for use by other machines, not just human actors. Dynamic discovery of (web) services will enable data residing in computers to be consumed dynamically with no previous knowledge of the provision of services. Having overviewed a concrete implementation of a relational database application exposed as XML, we now discuss other aspects of XML and relational structures.

C. XML AND RELATIONAL STRUCTURES

In a relational model a data item is stored as a field, a field within in record, and a record within a table. These data elements conform to a defined data type. The data item is indexed as appropriate and can be extracted using SQL. The same data item in an XML structure is stored as an element or attribute, and it is a child of a parent element and can conform to a data type (using an XML schema). It can be extracted using and Xpath query, a query language for XML.

One may erroneously conclude that one could simply surround relational data with element tags and call it XML. XML documents are designed to be human readable; that is, they provide human readable terms in the tag set to describe the data. However, at present, XML documents do not offer the flexible querying capabilities relational models provide. For example, Xpath queries require more processing than T-SQL[18] queries; thus they are slower. (Williams, 2002, 202) This situation results in at least one use case for exposing relational data as XML, and possibly executing T-SQL commands to

[18] T-SQL or Transact-SQL is the language used to administer Microsoft® SQL Server 2000™. Transact-SQL is an extension of the language defined in the SQL standards published by the International Standards Organization (ISO) and the American National Standards Institute (ANSI). (SQL Server Books Online)

manipulate data. Another benefit of exchanging data between applications using XML is the lower number necessary conversions. To exchange data between N possible formats would require N^2 conversion filters; whereas, using XML would require only 2N conversion filters (one for each direction from RDBMS to XML, and vice versa). (Birbeck, 2002, 40)

Mapping relational structures to other relational structures is relatively straightforward; however, cross-mapping between relational and XML is not as simple as a one-to-one mapping. For example, instances of the same element in XML are allowed. If this were allowed in a relational model, new tables with appropriate foreign keys would have to be constructed. Once the mapping has been defined, it is essentially fixed. Data exposed as XML loses the benefits of being stored relationally, but gains portability. Although XML is human readable, relational structures are not necessarily. A tradeoff between readability of XML documents and structured relational data must be weighed. When converting XML structures to relational databases, consideration must be given to the underlying relational model. If the relational database is new, this is less of an issue; otherwise, intermediate tables may be required. Despite these issues, the benefits of cross-mapping between XML and relational structures remain[19].

Various methods for effective, automatic conversion of XML data into and out of relational databases exist. All the major commercial RDBMS vendors such as IBM, Microsoft, Oracle, and Sybase have developed tools to assist in storing and exposing XML in relational tables.[20] The Route application implements the SQLXML API with SQL Server 2000. This technology offers the benefits described; however, there are limitations. For example, the SQLXML API currently does not implement the entire Xpath specification. (Williams, 2002, 208) Additionally, there are security implications to allowing URL access to data that must be considered. (Williams, 2002, 122) Having considered mapping between relational and XML models, we consider a new paradigm for storing XML.

[19] There are dangers if the terms being mapped are described in an ontology. While a mapping may occur on the surface (term to term) the underlying logic can conflict

[20] XML.com [http://www.xml.com/pub/a/2001/06/20/databases.html].

D NATIVE XML DATABASES

An alternative to storing XML in relational databases is the concept of a Native XML database (NXD). This solution offers advantages and disadvantages as compared to relational solutions. If the entire application deals with XML formats, there may be no need to convert between relational and XML formats; thus decreasing processing which increases application speed. Another benefit to XML databases is that many use the Xpath query language which is a more natural query language than SQL, but relegated to querying at the document level only.

A NXD is defined as[21]:

- Defining a logical model for an XML document – as opposed to the data in that document – and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, Parsed Character Data (PCDATA), and document order. Examples of such models are the Xpath data model, the XML infoset, and the models implied by the Document Object Model (DOM) and events in the Simple API for XML (SAX) 1.0.
- Has an XML document as its fundamental unit of logical storage, just as a relational database has a row in a table as its fundamental unit of logical storage.
- Is not permitted to have any particular underlying physical model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

Three key points from this definition are the following: The NXD is specialized for storing XML data and stores all components of the XML model intact. XML documents go in and XML documents come out. Third, a NXD may not be a standalone database. NXDs are not fundamentally a new database model, and they are not intended to replace relational databases. They are, however, a robust model for storage and manipulation of XML documents.

Though several implementations of NXDs exist, most are first generation. A small number of vendors currently implement second generation products. The NXD model is still maturing and will likely continue to do so. A casual review of the various XML databases reveals significant model variation across vendors.

[21] Definition taken from “Introduction to Native XML Databases” XML.com [http://www.xml.com/pub/a/2001/10/31/nativexmldb.html].

Despite the variances in implementations, NXDs store XML documents as a unit and create a model closely related to the XML or one of related XML technologies such as DOM. The fundamental NXD model includes arbitrary levels of nesting and complexity, as well as complete support for mixed content and semi-structured data. The input model is automatically mapped by the NXD to the underlying storage mechanism. The mapping used ensures the XML specific model of the data is preserved.

NXDs manage collections of documents, allowing one to query and manipulate those documents as a set. Collections basically are equivalent to tables in relational databases. XML documents are generically equivalent to tuples, or rows in relational models. NXDs diverge from the table concept in that not all native XML databases require a schema to be associated with a collection. This means an XML document can be stored in the collection, irregardless of a schema. Queries are still able to be constructed across all documents of the collection. NXDs that support "schema-less" functionality are termed *schema-independent* (or "non-validating"). Schema-independent collections give the database great flexibility and ease application development. Conversely, the risk of low data integrity increases. Strong schema support may be achieved by implementing a schema-dependent NXD or by including requisite document validation in the application design. For example, prior to writing data into the database, the data to be saved may be validated against a schema.

Xpath is the current NXD query language of choice. Xpath has been extended to allow queries across collections of documents. However, Xpath was not originally designed as a database query language and comes up short in several ways. Some limitations include a lack of grouping, sorting across document joins, and support for data types. Many of Xpath's limitations may be overcome by Extensible Stylesheet Language (XSLT), but a more database-oriented language is under development, called Extensible Query (XQuery)[22]. Several vendors have already released prototype XQuery implementations for use in their databases.

Updates are a significant weakness of NXDs. Most NXDs require the user to retrieve the document to be updated, change it with a selected XML API, and then return

[22] See [http://www.w3c.org/XQuery].

it to the database. The open source XML database named Xindice, implements XUpdate for this purpose; however, as of the time of this writing the XUpdate specification is still a work in progress, and is not fully functional.

1. Application Areas of NXDs

Other than the requirement to use XML, NXDs are quite flexible. For example, NXDs excel at storing document-oriented data such as XML and derivative technologies like XHTML, XSLT, or OWL. NXDs are likewise well-suited for data having a very complex structure and deep nesting, and data that are semi-structured in nature. In the context of the knowledge generation process, NXDs provide a rapidly deployable solution for enabled, semantic content with no mapping or interface issues to navigate.

NXDs store XML documents in collections which are readily accessible through HTTP. Ubiquitous and simple HTTP renders geography arbitrary. NXD collections essentially form a "web" of continually expanding and interwoven data structures. This complex, far-reaching structure will serve as a key technology enabler for the SWEB. That is, by interconnecting ontologies and semantic data structures identified with universal resource names software interpretable information will be widely available for machine consumption. This factor substantially will decrease the adverse effect of knowledge dispersion.

One of our research objectives was to experiment with XML databases to gain insight on how they perform compared to traditional relational databases. This was important to us as ontology documents and instances will most likely be in some from of XML. To that end, we developed two applications that use the open source NXD called Xindice.[23]

One application was a web application that allowed users to input and manage bibliography instances into Xindice. The instance documents stored in the Xindice database were bibliography references. The web application provided the common database facilities to add, delete, and update data in the database. However, as we employed Xindice, database instances were at the document level. All transactions had to retrieve the entire document to successfully complete. Further, Xindice is a non-

[23] See [http://www.xml.apache.org/xindice[for more information.

validating (i.e. schema-independent) database. As discussed earlier, this offers advantages and disadvantages. One advantage for our application included the ability to define an "external" schema that was easily modified and did not affect instance documents. Disadvantages included extra processing and increased potential for compromising data integrity. Extra processing stemmed from the need to retrieve the entire document to perform transactions. Greater potential for poor data integrity arose from the use of a non-validating database. In practice, the performance loss was largely unnoticeable, and integrity issues were addressed at data entry using a validating schema. In sum, we believe this was an application instance where either a relational or native XML database would prove equally suitable.[24]

Another application of Xindice included storing XML representations of OTH-GOLD text messages. In this application instance, semi-structured data messages (OTH-GOLD messages) were intercepted by a simple agent and parsed into a structured (XML) document. The XML instance document was then stored programmatically into a Xindice collection. Subsequently, another agent, upon notification, would "peer into" the appropriate instance document looking for relevant data, using Xpath to query the document(s). Depending on the results of the query, that agent would take specified actions, which might include parsing and storing the appropriate instance into another collection. The use of Xindice was well-suited for this application as it reduced processing and stored our data at its atomic level with no need for mapping to traditional data structures. Additionally, because our native data model was XML, it was simple to transform the relevant contents of the message into a higher level, semantic structure.

These two applications illustrate the point that no single solution is suitable for every use case; however, it did demonstrate utility for a native XML storage mechanism. Just as there is no single optimal choice for data storage models, there is not just one implementation of an NXD.

[24] Please note that formal tests, like scalability, were not conducted.

2. NXD Implementations[25]

There are approximately 20 different native XML databases on the market at this time. Some commercial products include SoftwareAG's Tamino, X-Hive, and Excelon. Open source implementations include Apache Xindice, eXist, and Ozone/XML. A list of NXDs and XML-enabled databases can be found at http://www.rpbourret.com/xml/XMLDatabaseProds.htm. Each product implements its own API, which increases the difficulty of developing applications using NXDs. Connecting to a native XML database is similar to connecting to relational databases that use JDBC or ODBC. The API used is called XML:DB and its purpose is to provide similar functionality. Each NXD that supports the XML:DB API must provide a database-specific driver that contains the database connection logic. Again, the driver concept is similar to JDBC and ODBC.

E. SEMANTIC STORAGE

Semantic representation is the goal of the Discovery and Generation phases of the knowledge generation process. The ability to discover described data with embedded meaning and structure them together as information is necessary to create knowledge. Much of the discussion has centered on transformation and representation of existing data sources; whereas semantic data sources imply a pre-existing semantic structure of data with meaning embedded as opposed to raw data. Information and data will be accessed and interpreted by machines and will provide the fuel for knowledge bases and inference mechanisms.

Representing data in a machine-readable format enables data to be "understood" by machines (i.e. agents). More concretely, native XML databases lend themselves readily to support the storage of semantic representation. Using namespace features to reference and link concepts and meaning between related documents effectively creates a web of meaning. The related documents may be reused or extended by applications arbitrarily and without regard to geography. The web effectively becomes monolithic and offers the potential for near automatic knowledge discovery.

[25] XML.com [http://www.xml.com/pub/a/2002/01/09/xmldb_api.html].

F. ENABLED DATA

From heterogeneous, semi-structured and structured data sources to highly structured, semantic representations, relevant data is transformed to a machine interpretable format, OWL. As previously discussed, relational data, or even semi-structured data such as USMTF messages, may be mapped to XML and then transformed to the OWL using XSL. Additionally, XML data may be stored natively in a Native XML database; thus reducing processing. Finally, relevant data may be stored from its inception in the semantic language of the system, OWL. Regardless of the source and its associated format, XML enables loose coupling of systems and semantic representation.

Data discovery and manipulation lead to storage. The storage facilities and methodologies discussed demonstrate a variety of options. Data may be stored using a variety of structured models; however, by definition, this structure will enable knowledge generation, or information usage. Prior to usage, however, the enabled, stored information must be transferred. One method for transferring information may be conducted through mobile autonomous agents. We argue that these agents are an extension of distributed computing, which is an extension of traditional client/server technology. To this end, we present a survey of the progression of distributed computing paradigms up to and including the Control of Agent Based Systems (CoABS) and SWEB services.

G. SUMMARY

It is only fitting to close with some perspective. There are no magic solutions. That is true of XML as well. The drawbacks of using XML with data include reduced speed and increased transmission time. The extra processing to get data to and from XML reduces the speed of applications. The larger file size of XML documents takes longer to transmit. (Hunter, 2001, 481) However, the flexibility, portability and other advantages realized far outweigh the added processing time. Additionally, it has been demonstrated that compression technologies can actually shrink an XML document to sizes less than their textual counterparts.[26] Further, decreasing latency of network operations will tend to diminish any notice of delays created by XML transmission.

OWL will be discussed in detail in subsequent chapters; however, it is worth noting at least one drawback to it. That is, although is fundamentally an XML technology, it is very intricate and complex. Presently, visual editors are scarce and are only at beta or "1.0" versions. With time this will change and thus simplify OWL development and deployment.

This chapter has discussed various aspects of data. We discussed the idea of unstructured, semi-structured and structured data. We then concentrated on structured data and illustrated some potential applications of XML, specifically related to the MVC paradigm, interoperability and the Semantic Web. We also discussed XML and relational database structures and how they might be used together. We also elaborated on Native XML database concepts and implementations. One point to take away from this chapter includes understanding the importance of representing data in such a way that it becomes machine-interpretable information. That is to say data is a building block of knowledge, and it is essential to effectively represent it semantically. With machine-readable information present we are able to develop agents capable of automatically, autonomously understanding the information and responding appropriately.

However, prior to elaborating on agents, it is necessary to discuss the methods agents will employ to access and "read" this stored information. These methods may be classified under the umbrella of "Distributed Computing." Effective distributed computing technologies and techniques factor importantly in connecting these "small worlds" of information repositories.

[26] In the 1999 Global Command, Control, Communications, and Intelligence Joint Warfare Interoperability Demonstration, an Air Tasking Order (ATO) message was compressed a smart compression utility. Dr. Robert Miller of MITRE reported that XML-based ATOs were actually smaller than the original ATOs in compressed MTF format (i.e., the compressed XML-ATO was 46KB with the smart compression utility; and the MTF-ATO was 72KB compressed with Pkzip).

IV. DISTRIBUTED COMPUTING: SMARTLY CONNECTING SMALL WORLDS

A. DISTRIBUTED COMPUTING

With data structured and able to be machine-processed and highly interoperable, we are ready to access and retrieve it. The purpose of this chapter is to demonstrate an effective means to interact with and across the network. Several distributed computing models are discussed and compared. These models culminate in current distributed computing paradigms and potential future ones. Our emphasis will be how the present and potential distributed computing mechanisms can best help to enable the SWEB.

Semantically enabled, structured data sources are a prelude to the processing and handling required to bring a data source to a usable state. After structuring data and subsequent representation as information, we must consider how to discover, interact, make appropriate requests, and rapidly move the results of our requests to the requestor (Edwards, 1999, 6). The "plumbing" or network infrastructure will impact how users/agents interact with information sources. We must not only be able to satisfy our requests for information, we must be able to do so in a loosely coupled,[27] flexible manner. Depending on the infrastructure or distributed computing model we implement, we benefit more or less. As with most things, there are tradeoffs and different approaches are not usually mutually exclusive. In network-centric warfare, the network is arguably the backbone; however, effective distributed computing will enable significant new possibilities. Certainly, in the Semantic Web (SWEB) domain it will be no less important.

The progression of distributed computing models has seen a number of designs, and just as XML has impacted data structuring and representation, it fulfills a vital role in a new distributed computing paradigm called "Web services." We believe Web services, more precisely, SWEB services will prove integral to improving workflow between organizations. To understand the SWEB services' role it is important to survey the progression of distributed computing. As such, we highlight some of the more prevalent

[27] By "loosely coupled" we mean that level of dependencies between systems or entities is low. Accordingly, changes in one system will not necessarily "break" the interaction between systems.

distributed computing models including Web services, which are presently achieving much notoriety in the distributed computing world. We then discuss SWEB services, an extension of Web services. The SWEB services model will be analyzed for utility and applicability to the SWEB, especially for military purposes.

In addition to SWEB services, the Control of Agent Based System (CoABS) implementation, an extension of Sun Microsystems® Jini™, will be discussed and analyzed in the context of agents and services. All these efforts are the latest in a broader concept of distributed computing known as Service Oriented Architecture (SOA). The aim of SOA is to enable software components, functions, objects and hardware devices on different systems to be accessible as services; that is, dynamic, ubiquitous, pervasive computing.

Our interest in distributed computing is largely centered on how to intelligently employ these emerging technologies, like CoABS/Jini™ and SWEB services, to speed the military decision maker's effective decision rate, thus gaining knowledge superiority. We envision agents interacting with semantic structures to assist with knowledge creation and decision making. Agents embedded with semantic structures for intelligence and constructed of mobile code for autonomy will be delegated the responsibility to acquire information needed in support of military operations. Prior to realizing these ambitious goals, we discuss the world of distributed computing in more concrete terms.

Distributed computing is a type of computing in which different components and objects comprising an application can be located on different computers connected to a network. Distributed computing involves computing on more than one computer system. Each computer in the distributed application has a role to play in the overall application. In a classic client/server model, a web client interacts with a remote application server which in turn may access a database server system; that is, a 3-tier architecture. This type of construct has morphed from the 3-tier architecture to a more sophisticated n-tier architecture where any number of components may combine to execute workflow processes. For example, consider logistics support to military operations wherein access to inventory management systems and ordering systems is crucial to operations support. Recent events in operation Iraqi Freedom highlight the importance of logistics support.

This (logistics) support is distributed across physical and logical boundaries, and across multiple organizations – military and civilian, foreign and domestic. The need for flexible and interoperable systems is readily apparent. It emphasizes the importance of establishing a robust, loosely coupled distributed computing mechanism. The ability to coordinate on such a grand scale and to orchestrate such logistics support is remarkable and unparalleled. Without the interconnecting infrastructure this feat would simply not be possible.

Other advantages of distributed computing include increased performance by applications working in parallel and spreading the load. Fusing intelligence from multiple sources and performing complex calculations using numerous processing devices are enhanced when distributed computing is employed. Collaboration among systems is also an advantage of distributed computing. The ability to share situational awareness information and a common operating environment substantially improves coordination and strategy development among military decision makers. Some key advantages are highlighted in Table 2.

Characteristic	**Description**	**Application**
Higher performance	Applications can execute in parallel and distribute the load across multiple servers.	Advanced numerical calculations, such as meteorological operations.
Collaboration	Multiple applications can be connected through standard distributed computing mechanisms.	Shared situational awareness and common operating environments.
Higher reliability and availability	Applications or servers can be clustered in multiple machines.	Supports redundancy in case of system failures.
Scalability	This can be achieved by deploying the reusable distributed components on powerful servers.	Facilitates advanced Research and Development, Testing and Evaluation.
Extensibility	This can be achieved through dynamic (re)configuration of applications that are distributed across the network.	Supports ability for mobile units to maintain connectivity through changing areas of operations.
Higher productivity and lower development cycle time	By breaking up large problems into smaller ones, these individual components can be developed by smaller development teams in isolation.	Facilitates systems design and development.
Reuse	The distributed components may perform various services that can potentially be used by multiple client applications. It saves repetitive development effort and improves interoperability between components.	Allows multiple organizations to interact with one another. Facilitates interoperability between Services and Agencies.
Reduced cost	Because this model provides a lot of reuse of once developed components that are accessible over the network, significant cost reductions can be achieved.	Reduces life-cycle cost of systems.

Table 2. Advantages of Distributed Computing (After: Nagappan, 2003, 5).

After considering the advantages of distributed computing and it significance, we survey the progression of distributed computing.

1. Client/Server Architecture

In the early, 2-tier, systems there was an upper layer and a lower layer. The upper layer contained the presentation and business logic (client), and the lower layer encapsulated the application organization and data storage (server). In the client/server scheme the server is generally a database server and organizes and retrieves data. The client contains the Graphical User Interface (GUI) logic and executes the business logic. This was brittle because changes in the server necessitated changes in all the clients. This high-maintenance model was costly and impractical, and led to 3-tier systems wherein the business logic and other control logic were executed on the server in a middle layer between the client and the application data. One primary advantage of this was to essentially decouple the client from the application. Generic clients could be built to interact with multiple applications. Examples of client/server systems abound. Web-based applications almost exclusively implement the client/server structure. From the intelligence community to logistics agencies collaborating and sharing data, and to individual military units advertising local amenities, this structure dominates. However, the client/server approach it is not perfect.

A balance between server-side and client side processing is always sought. Client-side processing requires robust (fat) clients to execute application-specific logic. Placing more logic on the client effectively speeds the server; however, more coupling occurs as the client must be more and more application-logic savvy. Conversely, increased processing on the server leads to more “thin” clients and looser coupling; however, it effectively slows down the server. The need to balance the load is an ongoing exercise, and design decisions invariable affect pre-existing applications, and applications to come. Client/server systems are vulnerable to hacking. Increased network bandwidth demands result from calls to the server. These are typically database oriented applications.

2. CORBA

Remote Procedure Call (RPC) is a method for executing functions/methods on remote machines.[28] Similar, but incompatible RPC standards include the multi-platform Common Object Request Broker Architecture (CORBA) and the Microsoft-specific Distributed Component Object Model. (Birbeck, 2001, 41) CORBA is based on open standards, and was developed by the Object Management Group (OMG).[29]

CORBA is different from client/server systems in that it is an object oriented solution. It is not based on proprietary solutions. CORBA is programming language, operating system, and platform agnostic. CORBA operates by mapping specific languages to a common interface language called Interface Definition Language (IDL). IDL is designed to expose services (methods/functions) of a CORBA remote object.

In the CORBA model, disparate languages such as Java, C, and C++ communicate with a neutral IDL and interoperate via a Common Object Bus. The Object bus provides the communications infrastructure to send and receive requests/responses between clients and servers. It is the foundation that enables interoperability in a heterogeneous environment. Figure 17 below illustrates an example architectural model of CORBA using different programming languages. The IDLs between the server skeletons and the server classes provide interfaces for calling applications (clients) using a standard language supported by the ORB object bus.

[28] [http://www.xmlrpc.com], 4/23/2003.

[29] OMG is a non-profit consortium responsible for production and maintenance of framework specifications for distributed and interoperable object-oriented systems.

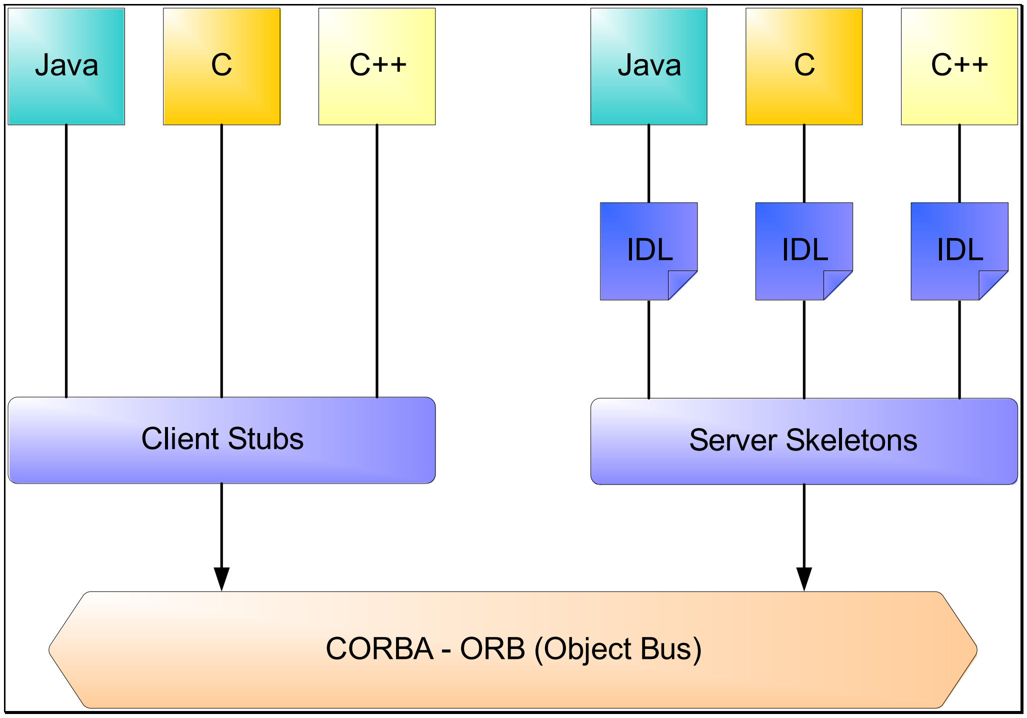


Figure 17. Example of CORBA Architectural Model (From: Nagappan, 2003, 9).

Advantages of CORBA over client/server architectures include the following.

- Operating System and programming language independence
- Legacy and custom application integration
- Rich distributed object infrastructure
- Location and network transparency
- Remote callback support. Objects may receive asynchronous event notification from other objects.
- Dynamic Invocation Interface. CORBA clients can use static or dynamic method invocations. They can define method calls at compile time or discover objects' methods at runtime.

Despite these advantages there are success/adoption inhibitors. For example, there are high initial investment costs in the form of training and coding. Availability of CORBA services is limited to a small number of implementations. Scalability is limited due to the tightly coupled nature of the connection oriented architecture. The next distributed computing model we discuss is (Java) language-centric; that is, Remote Method Invocation.

3. Java RMI

The Java Remote Method Invocation (RMI) was developed by Sun Microsystems. RMI enables distributed Java object-based application development using the Java programming language. RMI extensively uses java object serialization which is a technique that allows objects to be converted to data streams, which are readily transported over network communication protocols. RMI uses the Java Remote Method Protocol (JRMP) as the inter-process communication protocol, enabling Java objects residing in different Java Virtual Machines (JVM) to transparently invoke one another's methods. Because the JVM can be on different machines anywhere on the network, RMI enables object-oriented distributed computing.

RMI employs a registry mechanism to instantiate a "lookup service." The lookup service stores references to remote objects and enables object lookups from client applications. The RMI infrastructure acts as a medium between RMI clients and remote objects. Figure 18 illustrates the basic interaction between RMI components. Essentially, the client object makes calls on the remote server via the RMI stub. The RMI stub serves as a proxy representing the remote service. The stub uses the RMI protocol to communicate back to the RMI skeleton over the network. The RMI skeleton interacts with the actual remote services to execute the actual method calls, and then returns the results.

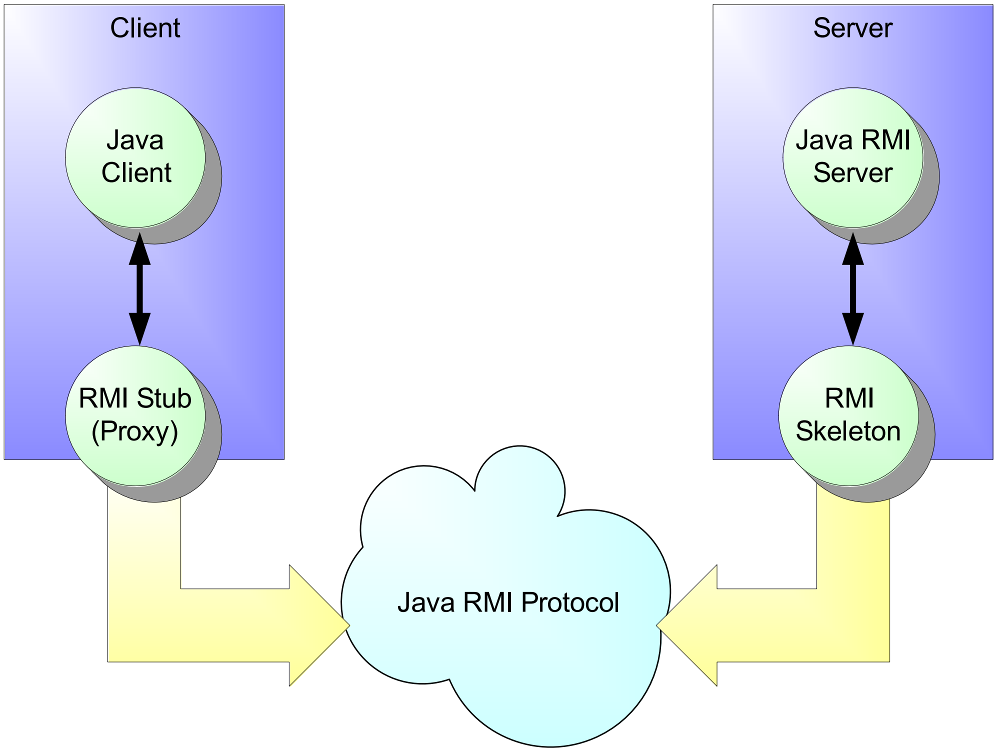


Figure 18. RMI Client/Server Communication (After: Edwards, 1999, 724).

The RMI architecture is composed of the components described in Table 3.

Component	Description
RMI client	Performs the remote method invocations on a server object. It can pass arguments that are primitive data types or serializable objects.
RMI stub	The client proxy that encapsulates the network information of the server and performs the delegation of the method invocation to the server. The stub also marshals the method arguments and unmarshals the return values from the method execution.
RMI infrastructure	Consists of the remote reference layer and the transport layer. The remote reference layer separates the specific remote reference behavior from the client stub. The transport layer provides the networking infrastructure, which facilitates transporting data to and from method invocations.
RMI skeleton	Receives method invocation requests from the stub and processes the arguments (unmarshalling) and delegates them to the RMI server. It also marshals the return values and then passes them back to the RMI stub via the RMI infrastructure.
RMI server	The Java remote object that implements the exposed interfaces and executes the client requests. It receives incoming remote method invocations from the appropriate skeleton, which passes the parameters after unmarshalling. Return values are sent back to the skeleton, which oases them back to the client via the RMI infrastructure.

Table 3. Java RMI Components (After: Nagappan, 2003, 12).

RMI frees programmers from developing application-level protocols necessary for encoding and decoding messages for data exchange. RMI interoperates with CORBA components through RMI-IIOP (Internet Inter-ORB Protocol). Fundamentally, RMI is a Java-programming-language-enabled extension to traditional RPC mechanisms. RMI not only allows data to be passed between objects over the network but also full objects, including code. (Jini™ AO, 1999, 7)

RMI is limited to the Java platform; there is no provision for language independence. Application architectures are tightly coupled because of its connection-oriented nature, making scalability difficult to achieve. There is no provision for specific session management support.

The next model is a platform-specific solution called DCOM. It is somewhat similar to RMI in that it is used in a Windows computing environment; whereas, RMI is a Java based solution, but can be run on various platforms and operating systems.

4. Microsoft™ DCOM

The Microsoft Component Object Model (COM) provides a way for Windows-based software components to communicate with each other by defining a binary and network standard in a Windows operating environment. COM provides a distributed application model for ActiveX components. The follow-on technology to COM is Distributed COM (DCOM). DCOM is Microsoft's answer to distributed computing in a Windows environment. DCOM allows COM objects to communicate via a Remote Procedure Call (RPC) mechanism. Similar to RMI, DCOM uses a stub and skeleton approach whereby a defined interface that exposes COM object methods can be invoked remotely over a network. DCOM servers can host multiple COM objects, and when they are registered in a registry, they become available to all clients who discover them using s lookup mechanism.

Similar to RMI, DCOM is quite successful in distributed computing support in a single platform environment, namely Windows. DCOM limitations are essentially the same as RMI; they include language lock-in, difficult scalability, and complex state and session management issues.

5. Message Oriented Middleware

CORBA, RMI, and DCOM share a common, tightly coupled, synchronous communication model (i.e. request/response). This tight coupling leads to scalability issues and brittle application implementations. Message Oriented Middleware (MOM), also known as message-queuing systems, is based on a loosely coupled, persistent asynchronous communications model where a client may be ignorant of its application recipients or its method arguments. MOM enables applications to communicate indirectly using a message provider queue.

In a MOM architecture, the client sends messages to a message queue (essentially a message buffer), and the receiving application picks up messages from the queue. The message-queuing system provides persistent communication by storing messages as long as it takes to deliver them to the receiver(s). (Tanenbaum, 2002, 100) In this construct, the message sending application continues to operate without waiting for a response from the message receiving application. An important aspect of message-queuing systems is that a sender is generally given only the guarantee that its message will eventually be inserted in the recipient's queue. No guarantees are given about when, or even if the message will actually be read, which is completely determined by the behavior of the recipient. These characteristics enable the loosely coupled communications facility. (Tanenbaum, 2002, 109) The message queuing system provides queues for senders and receivers, and is responsible for managing those queues. To ensure or increase reliability, messages may be persisted in a database or file system. The message queues are distributed over multiple machines; therefore, for the message queuing system to properly transfer messages it must implement a mapping of queues to their respective network locations. This mapping is completely analogous to the Domain Name System for e-mail in the Internet. (Tanenbaum, 2002, 111)

The asynchronous messaging paradigm implemented by MOM systems is comparable to e-mail systems used by human actors. E-mail sent between users may be stored in the recipient's inbox for an indefinite time before being read and acted upon. In the meantime, the sender is free to send other e-mail messages or engage in any other activity. At a later date, should the recipient reply to the sender, a transaction may be completed or may simply lead to additional message exchanges. In much the same paradigm, software applications employing asynchronous messaging are free to continue execution while a message goes unanswered.

Some implementations of Message Oriented Middleware include Sun ONE Messaging Queue, IBM MQSeries, TIBCO, SonicMQ, and Microsoft Message Queue (MSMQ). Java provides the Java Messaging System (JMS) API. XMLBlaster is an open source MOM implementation.[30] Some challenges of MOM are native APIs which lead to vendor lock-in, and proprietary message formats.

6. Jini™

RPC systems attempt to make the call of a function on a different machine look (to the programmer) like the call of a local function in the same address space. Remote object systems like CORBA and DCOM raise the level of programming from function calls to method invocations on objects but still essentially try to mimic the semantics of local invocation. All of these systems, including XML Web services, try to make the network transparent or "go away," from the programmer's perspective. But the network is not transparent. The hardest parts about developing distributed systems are the aspects that do not "go away." For example, the time required to access a remote resource (latency) may be orders of magnitude greater than accessing the same resource locally; networks fail in ways standalone systems don't; and networks are susceptible to partial failures of computations that can leave the system in an inconsistent state. RPC and CORBA don't even consider performance and latency as part of their programming models – they simply ignore the issue entirely. (Edwards, 1999, 43) New failure modes require that the system not only know what to do when an error occurs, but that it also be aware an error occurred in the first place. Additionally, failures within distributed systems – from a single software process to an entire computer failure – "somewhere" on the network can leave the system in an inconsistent state.

[30] XMLBlaster software is written in Java and is free for private, commercial, and educational use. (See [www.xmlblaster.org]).

An n-tier network is designed in such a way as to enable the services to interact with each other within the network. The technologies used to solve this problem are CORBA, DCOM and other RPC-based systems. These technologies must:

- Know about each other while being built, at least to the extent of sharing IDL definitions
- Be developed in lock step, because changing the stub or skeleton files requires changing all of the corresponding stub or skeleton files.
- Require fairly tight levels of administration

Jini™ is an attempt to solve these problems. The Jini™ vision is this: When you walk up to an interaction device that is part of the Jini™ system, all of its services are as available to you as if they were on your own computer – and services include not only software but hardware devices as well; that is, almost anything imaginable that passes information in and out. Adding a new device to Jini™ is as simple as plugging it in. Jini™ systems will combine computer and consumer devices along with a variety of external sources – the Internet, broadcast, cable, satellite and landline. (Jini™ Now?, 1999, 2)

To help realize this vision, Jini™ makes assumptions about distributed systems that other approaches do not consider. Computer scientist Peter Deutsche, who worked at Sun Microsystems in the early 1990s, coined a set of distributed computing fallacies known as Deutsche's Eight Fallacies of Distributed Computing (based on problems that happen in real networks over time). The following eight fallacies provide a back drop from which to consider Jini™ characteristics.

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

Jini™ can deal with inherent unreliability of the network; keep up with constantly changing topology; allow multiple administrators; evolve components; take failure seriously in the construction of interactions. Jini™ extends the Java application environment from a single virtual machine to a network of machines. The Jini™ system federates computers and computing devices into what appears to the user as a single system. It is assumed that the latency of the network is reasonable.

It assumes that each Jini™ technology-enabled device has some memory and processing power. Devices without processing power or memory may be connected by a software or hardware proxy that contains processing power and memory. (Jini™ AO, 1999, 3)

Jini™ technology consists of an infrastructure and a programming model that address the fundamental issue of how clients connect with each other to form an impromptu community. Jini™ lets programs use services in a network without knowing anything about the wire protocol used by the service. One service might be implemented using an XML-based protocol; another might be RMI-based, or CORBA-based. Jini™ technology is often compared to universal plug and play (UPnP), which is a discovery protocol for devices. Jini™ technology and UPnP, or any connectivity scheme, can interoperate because the Jini™ architecture is wire-protocol and transport-protocol neutral. (Jini™ EO, 2001, 14) Jini™ adds a number of incremental changes to Java to extend it to create a lightweight, distributed computing architecture. Based on the Java language, Jini™ technology uses the methods incorporated by Java RMI protocols to move objects, including their behavior, around the network. Network services run on top of Jini™ software.

Figure 19 depicts one conceptual use of Jini™ and illustrates the various protocols used within the Jini™ technology architecture. In this use case, a terminal access sensor data via a Jini™ service and also accesses and/or writes to a knowledge repository (KB). This particular example shows one service (satellite sensor) using a device protocol over the Jini™ protocol. The other service (knowledge repository) is accessed via an alternate protocol, such as an XML Web service. The knowledge repository could also be a semantically-enabled information source that might allow the decision maker to realize new information based on the incoming sensor data and the information resident in the knowledge repository. Depending on the application, new sensor data may be combined with the represented knowledge in the KB to adapt the domain model.

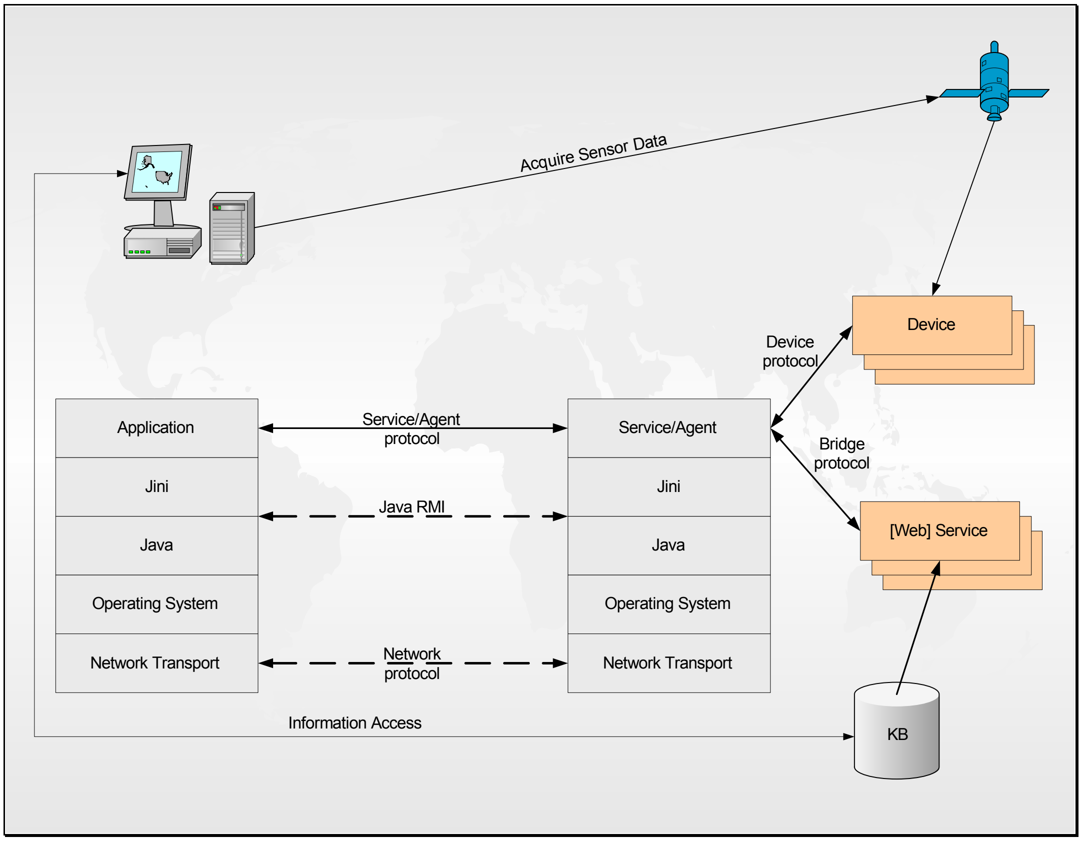


Figure 19. Jini™ Conceptual Usage (From: Jini™ TE, 1999, 1).

Java, although not a panacea, makes some strong progress on the issues that increase the difficulty of programming distributed systems. First, Java assumes the existence of a particular language everywhere; whereas, systems like CORBA and DCOM are bolted on accessories to one or more languages. Java code is portable; and thus readily mobile, and therefore, is able to move to the data to operate. The mobility feature of Java makes it a prime candidate for implementing autonomous agents. Java code moves as easily from machine to machine as data moves, even machines with different operating systems or CPUs. Therefore, agent code can traverse the net, gaining performance by moving close to the required data. The Java bytecode format is the universal interchange format for executable code. (Edwards, 1999, 47)

Java's security code mechanism ensures that code, once moved into a system, can execute with a greater degree of safety than other approaches. For example, Java allows fine-grained access control to machine resources based on security policies. (Edwards, 1999, 47) The distributed security model and its implementation define how entities are identified and how they get the rights to perform actions on their own behalf and on the behalf of others.

Again, Java is not a universal remedy, and it does not provide miracle solutions to the problems described earlier; however, its model of distributed computing explicitly acknowledges the differences in local and remote computing and provides tools to help deal with them. (Edwards, 1999, 48) Additionally, one of Java's strengths is its APIs supporting networking.

RMI makes the problems of data portability simply go away. (Edwards, 1999, 52) RMI defines the base language within which the Jini™ services communicate. And, by supporting secure and safe code motion, RMI allows the benefits of object oriented programming to communicate between objects across device boundaries, and to work across the network boundary. RMI provides mechanisms to find, activate, and garbage collect[31] object groups. Much of the Jini™ system is enabled by the ability of RMI to move code around the network in a form that is encapsulated in an object. Jini™ leverages RMI and uses mobile code as a way to achieve maintenance, evolvability and ease of administration for networked devices and services. RMI enables activation of objects and the use of multicast protocol[32] to contact replicated objects. Jini™ is not RMI. Jini™ uses RMI extensively, particularly its facilities for mobile code. Jini™ is a set of services and conventions built on top of RMI. In fact, the RMI Activation framework includes an RMI Daemon (RMID) that, along with the Lookup Service, stores information in logs. This is a characteristic as it ensures that they can continue from where they stopped in the event of a machine or process failure. The RMID will

[31] Java performs automatic garbage collection of memory to help return memory back to the system. When an object is no longer used in the program (i.e., there are no references to the object), the object is marked for garbage collection. The memory for such an object can be reclaimed when the garbage collector executes. (Deitel, 2002, 426).

[32] Multicast protocol is similar to broadcasting, wherein a sender sends to multiple receivers without discrimination. Multicasting differs in that it does not send to every node on the network, but only to interested nodes. For a more complete discussion of Multicast protocols, see Kumaran, 55-56.

remember how to activate the Lookup Service, and the Lookup Service will retain the contents of its registry when the RMID is restarted. (Kahn, 2002, 10) Because Jini™ is layered atop RMI it takes full advantage of the Java language, and the other benefits of mobile code.

Jini™ has similarities to the Internet's Domain Name Service (DNS); it even provides a service for finding other services in a community. But Jini™ differs from DNS in that it supports serendipitous interactions among services; that is, Jini™ allows services to appear and disappear without the need for static configuration or administration. This feature is termed "spontaneous networking." The other important difference is that services "close"[33] to one another form a community automatically, without the need for human intervention. Notably, the ability to operate across LANs is limited due to firewall considerations. In a controlled environment, however, this should prove to be incidental. That is, the interacting communities of interest will be in one logical network from a security perspective.

Communities of Jini™ services are self-healing – a key property built into Jini™ from the ground up. Jini™ makes the assumption that networks and software fail over time. Given time, the system will repair itself. Jini™ also supports redundant infrastructure, reducing the possibility that services will be unavailable if key machines fail. (Edwards, 1999, 59) The RMI Activation Framework provides a way for objects to be automatically reconstituted from persistent storage.

These properties serve to reduce substantially the administration required for a Jini™ community. Spontaneous networking means no manual configuration of the network. Additionally, the ability to find and use devices previously unknown means there is no need to install drivers or software. As presently designed, prior knowledge of Jini™ service's interface is essentially the only parameter that must be known prior to using it.

Jini™ addresses scalability through federation. Federation is the ability for Jini™ communities of services to be linked together into larger groups – federations.

[33] "Close" refers to LAN segments addressed. The multicast protocol used for service discovery may be adjusted to alter the number LAN segments that are addressed; this effectively increases (or decreases) the scope of service discovery.

Importantly, Jini™ is device-agnostic; that is, only the interface to the device needs to be understood. In fact, the device or service does not need to be written in or understand Java. Any programming language can be supported by a Jini™ system if it has a compiler that produces compliant bytecodes for the Java programming language. (Jini™ AO, 1999, 3)

Jini™'s ability to create spontaneous, self-healing communities of services is based around five key concepts: Discovery, lookup, leasing, remote events, and transactions.

Discovery is the process used to find communities on the network and join with them. Dynamic discovery is responsible for spontaneous community building.

Lookup fulfills the role of a directory service within each community, and provides facilities for searching and finding services. The lookup service reflects the current members of the federation and acts as a central marketplace for offering and finding services by members of the federation. It is more complex than a simple name server that maps strings onto objects. The lookup operation can search based on the type of object, and can consider inheritance relationships during the search. For example, if one searches for "ground intelligence services," the search may return both "supersets" of ground intelligence services and "subsets" of ground intelligence services, and of course ground intelligence services. As well, searches may be narrowly defined to return only ground intelligence services. Services are found and resolved by a lookup service. The lookup service provides the major point of contact between the system and the users of the system. A service is added to a lookup service by a pair of protocols called discovery and join – first the service locates an appropriate lookup service (using the discovery protocol), and then it joins it (using the join protocol). The discovery/join protocol defines the way a service becomes part of the Jini™ system. Many active lookup services help increase fault tolerance of the system.

Leasing is used extensively in Jini™. A lease is a grant of guaranteed access over a period of time. Leasing is the technique that provides Jini™ with its self-healing nature. Leasing ensures that a community will, after a time, recover the loss of any key services. If a service goes away, intentionally or not, its leases eventually expire and the service will be forgotten. Access to many of the services in the Jini™ system is lease-based. The lease interface defines a way of allocating and freeing resources using a renewable, duration-based model. Leasing enables references to objects to be reclaimed safely in the event of network failures. Jini™ makes extensive use of logs to checkpoint service states periodically and to recover after a crash.

Remote events allow services to notify each other of changes of state. Service state change notification events to clients may be asynchronous. Jini™ supports remote event notification that enables event-based communication between services.

Transactions are a mechanism that allows computations that involve multiple services to reach a safe state. That is, computations either completely succeed or are not completed at all. This trait helps mitigate partial failures. Jini™'s transaction model resembles the classical database transaction model.

We have referenced the notion of services throughout this section, but what is a Jini™ service? A (Jini™) service is an entity that can be used by a person, a program or another service. A service may be a computation, storage, a communication channel to another user, a software filter, a hardware device, or another user. More interestingly, a service may be an agent performing tasks on behalf of another agent or a human. In the military domain, a service may be a ship that is able to transit into and out of Jini™ lookup Services automatically with no reconfiguration necessary. Likewise, a lookup service may store object references to multiple ships, or multiple services offered by various ships or other entities. Conversely, a service may be simply a printer, or handheld device. Services are scalable in that they may represent the capabilities of a military unit or platform, all the way "down" to a storage device or software agent. Services appear as Java objects, perhaps made of other objects. These objects are proxies that represent the actual service. The proxy may implement the service entirely (i.e., self-contained service). Or the proxy may represent the service and communicate back to the actual service via some protocol – not necessarily RMI. As Figure 20 below indicates, a service item object captures the components that comprise a service; that is it contains an interface (or proxy) which defines the operations that can be requested of that service, a unique service identifier, and attributes that describe the service. (Jini™ AO, 1999, 12)

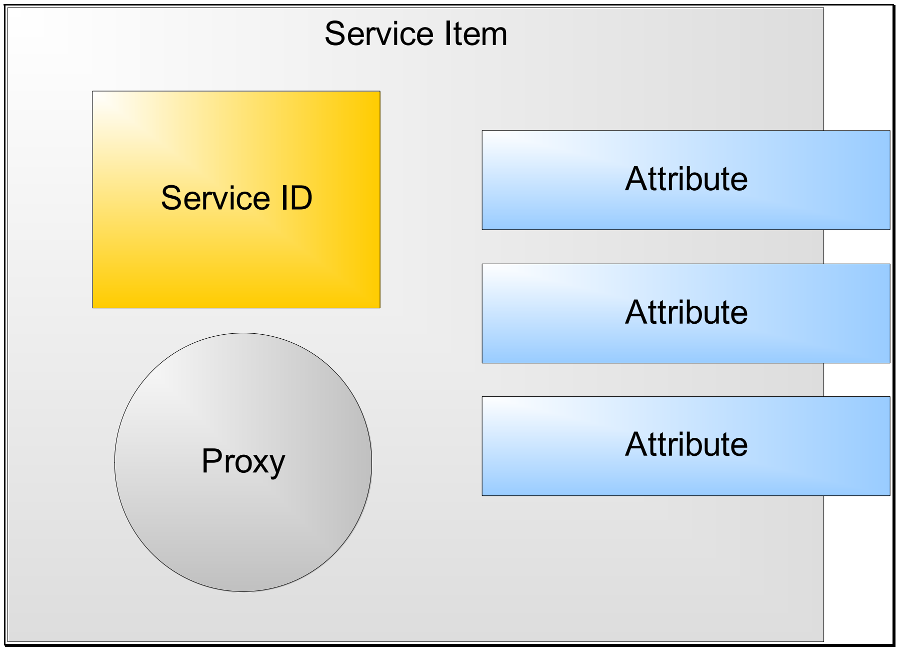


Figure 20. Components of a Jini™ Service Item (After: Edwards, 1999, 70).

Services in Jini™ are implemented using well known interfaces. This is the only way services can know *how* to programmatically interact with each other. Using standard interfaces ensures Jini™ services can take advantage of each other. Because services expose interfaces, developers can design to the service interface, and not to each protocol. Because interfaces are used to determine how to interact with a Jini™ service, the implementation of the interface may change without "breaking" the larger interaction between the service and the consuming clients. This characteristic provides a degree of decoupling while promoting standard interface designs.

Although Jini™ is not a Web service[34] in the conventional sense, it can communicate with Web services, or be used to "house" or build Web services. In fact, Web-enabled services will cause enterprises to run into the problems that are addressed by Jini™ technology. Some of these problems include:

- Finding and connecting services on a network
- Creating reliable sets of services out of unreliable parts, including an unreliable network

[34] Web services are discussed later in this chapter.

- Dealing with networks that are very large or long-lasting
- Evolving parts of the service set without halting the service set itself at any time
- Explosive growth in bandwidth, networks and digital devices is leading to state where Web services will become smart because they are context-aware and can be reliably delivered over multiple networks. Jini™ comes into its own when participants within a network start looking for services not just from other networks, but from the participants in the other networks as well. (Jini™ EO, 2001, 9) Extensive use of the network to execute workflow will inevitably lead to the occurrence of more failures in the network. These failures will often be partial failures in that the failure will cause a degradation that impacts a service, but doesn't necessarily shutdown a service. Jini™ provides a unique ability to handle partial failures by requiring services to implement facilities that explicitly raise remote exceptions, objects that allow a Java program to intelligently respond to that failure. Jini™ is an instance of a Service Oriented Architecture; it is protocol independent; supports dynamic community formation and dissolution; is location independent; is self-healing; and it is dynamic and self-managed. (Kumaran, 2001, xvi)

One of the seeming misconceptions of Jini™ some have is thinking of it as a Web service. This line of reasoning leads one to ask why Jini™ has not gained traction as a Web service. As stated earlier, it is not a Web service per se. We see utility for Jini™ and Web services working together. This idea is mentioned is the Web services section of this chapter.

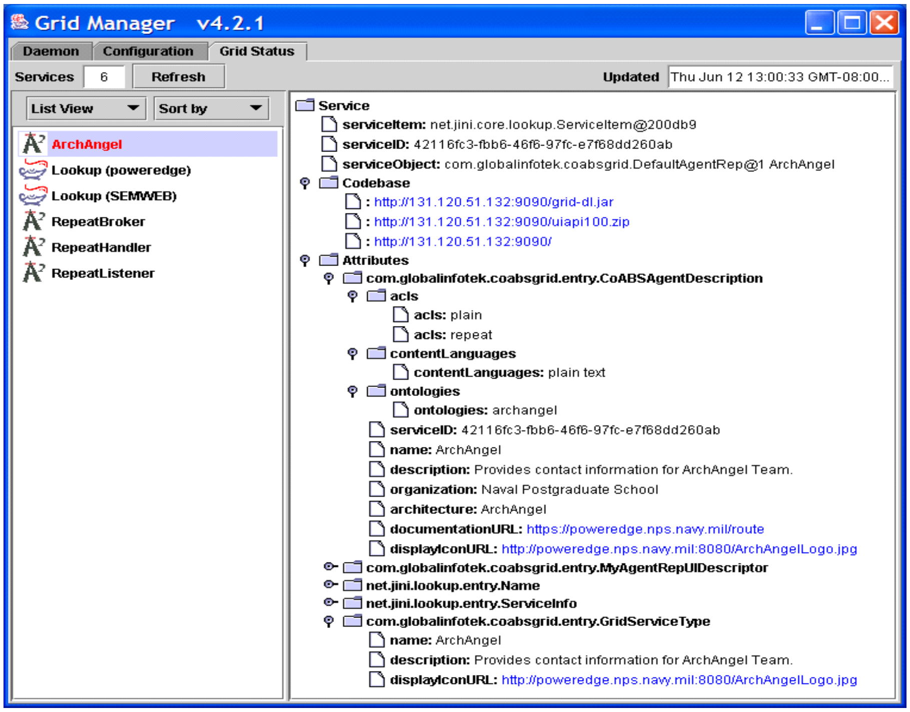
As mentioned earlier, Jini™ is well-suited for development and deployment of autonomous agents; however, the Control Agent Based System (CoABS) is an extension of Jini™ that not only manages services, it and provides an agent-based platform. As such, discussion of CoABS will focus on the agent aspect of services. Additional information about Jini™ can be found at http://www.jini.org and http://www.javasoft.com/products/jini.

7. CoABS Grid

Selection of CoABS as an agent platform for our research was a logical choice for us as it is closely related to Jini™.[35] This helped reduce the learning curve and allowed

[35] Other potential agent platforms exist. For example, JXTA (short for "juxtapose", as in side by side) is a set of open, generalized peer-to-peer protocols that allow any connected device on the network to communicate and collaborate. (See[http://www.jxta.org]).

us to develop agents more quickly. The Control of Agent Based Systems (CoABS) Grid is an extension of Jini™. It is built on top of the core functionality provided by Jini™. The CoABS Grid Users Manual defines the CoABS Grid as middleware that integrates heterogeneous agent-based systems, object-based applications, and legacy systems.[36] It also indicates that the CoABS Grid includes a message-based API to register agents, advertise their capabilities, discover agents based on their capabilities, and send messages between agents. CoABS Grid, being built on Java, enjoys the ability to execute mobile code on heterogeneous platforms. Mobility is a key characteristic of a software agent. Additionally, sending messages between agents, an implementation of persistent asynchronous communications, enables loosely coupled interactions between and among organizations of agents – another necessary characteristic of agents. Figure 21 is a sample screenshot of the CoABS grid user interface. The pane depicted in the figure is the Grid Status pane. It lists all agents and services registered in the local Lookup Service, including Lookup Services themselves. In this instance, two Lookup Services (SEMWEB, poweredge) are registered, along with four agents from the ArchAngel project.



Services 6

Updated Thu Jun 12 13:00:33 GMT-08:00...

ArchAngel
Lookup (poweredge)
Lookup (SEMWEB)
RepeatBroker
RepeatHandler
RepeatListener

```
Service
  serviceItem: net.jini.core.lookup.ServiceItem@200db9
  serviceID: 42116fc3-fbb6-46f6-97fc-e7f68dd260ab
  serviceObject: com.globalinfotek.coabsgrid.DefaultAgentRep@1 ArchAngel
Codebase
    : http://131.120.51.132:9090/grid-dl.jar
    : http://131.120.51.132:9090/uiapi100.zip
    : http://131.120.51.132:9090/
Attributes
  com.globalinfotek.coabsgrid.entry.CoABSAgentDescription
    acls
        acls: plain
        acls: repeat
    contentLanguages
        contentLanguages: plain text
    ontologies
        ontologies: archangel
    serviceID: 42116fc3-fbb6-46f6-97fc-e7f68dd260ab
    name: ArchAngel
    description: Provides contact information for ArchAngel Team.
    organization: Naval Postgraduate School
    architecture: ArchAngel
    documentationURL: https://poweredge.nps.navy.mil/route
    displayIconURL: http://poweredge.nps.navy.mil:8080/ArchAngelLogo.jpg
  com.globalinfotek.coabsgrid.entry.MyAgentRepUIDescriptor
  net.jini.lookup.entry.Name
  net.jini.lookup.entry.ServiceInfo
  com.globalinfotek.coabsgrid.entry.GridServiceType
    name: ArchAngel
    description: Provides contact information for ArchAngel Team.
    displayIconURL: http://poweredge.nps.navy.mil:8080/ArchAngelLogo.jpg
```

Figure 21. CoABS Grid Graphical User Interface.

[36] The CoABS Grid is only part of the overall CoABS program [http://coabs.globalinfotek.com/].

The CoABS Grid is the infrastructure that connects components together. The Grid can also be thought of as the infrastructure layer and all the agents and services running on it. (Kahn, 2002, 15) As stated, CoABS Grid is built using Jini™; however, the Grid provides helper utility classes that are local to an agent and that shield the user from the complexity of Jini™. (Kahn, 2002, 15) The programmer is still able to use Jini™ classes directly if desired. The concept of a service is represented as an agent in CoABS. In fact, agents and services may be developed in CoABS; the difference is that services are method-based; whereas agents are message-based. More specifically, clients engage and interact with services by making calls on the service's available methods. Whereas, agents send messages to each other using the uniquely identifying attributes, such as the agent's "name." Similar to the search for services described earlier, if one desires to find a variety of agents satisfying stated criteria, one may initiate a search accordingly and all available agents matching the criteria will be returned. The calling agent is then able to interact with one or more of the agents returned. Similarly, messages can be sent to a group of agents. The Grid uses the Jini™ Lookup Service to register and discover agents - and services.

The Grid is transport neutral in terms of agent communication. RMI is the default transport for messages. Message queues are implemented to store messages and (message) listeners are implemented to automatically notify agents of incoming messages. Although several classes of messages are provided in the Grid, some of which contain text only while others may contain data attachments, the Grid does not specify a language for agent communication. Developers must implement preferred agent communication languages. (Kahn, 2002, 17)

A primary goal of the Grid is to integrate agent, object, and legacy systems. (Kahn, 2002, 20) Integrating legacy systems is one of the main benefits of using the Grid. (Kahn, 2002, 55) Legacy code can be wrapped using the Grid classes. Moreover, the use of Java and XML as underpinning technologies enables greater interoperability with legacy systems.[37]

[37] The notion of legacy systems may fade into history as XML- and Java-related technologies proliferate.

In summary, the CoABS Grid extends Jini™ technology and seeks to reduce complexity. Using the Java programming language and the RMI Activation Framework, the Grid helps us to realize the concepts of autonomous, software agents. Figure 22 illustrates how these technologies build upon each other. Java provides mobility and flexibility; Jini™ provides spontaneous networking, and automatic discovery of self-healing agents and services; and asynchronous messaging provided by the CoABS Grid decouples processes, and supports autonomy and communications. Using the CoABS Grid, agent architectures may be created and enhanced to form intelligent agent societies. The ability for agents or services to enter and exit from the architecture with no human intervention also contributes to autonomy, and reduces, or arguably eliminates, the need for system administration.

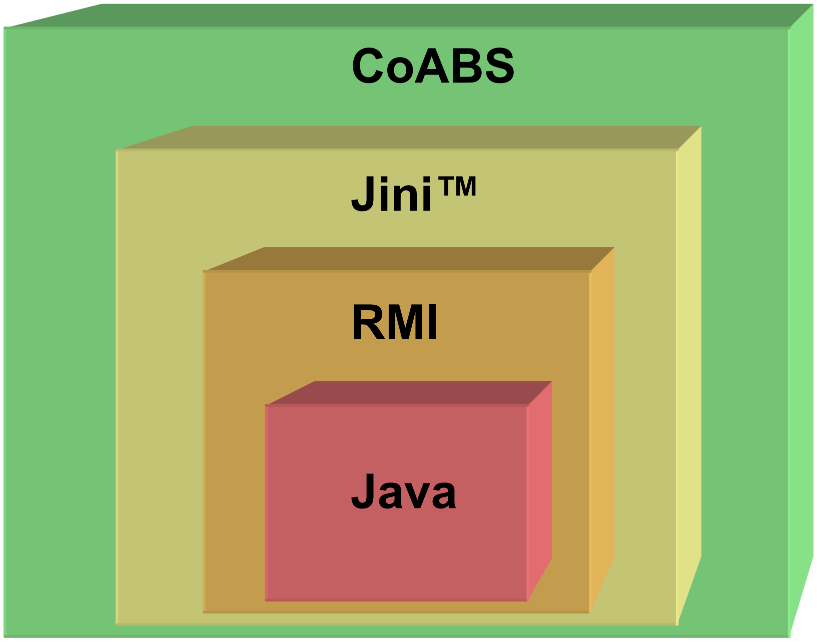


Figure 22. CoABS in Relation to Other Technologies.

8. Challenges in Distributed Computing

- CORBA, RMI, and DCOM are successful at integrating application in a homogeneous environment within a Local Area Network. Cross-network solutions are necessary.
- Maintenance of stubs and skeleton is extremely complex in a heterogeneous environment.

- Quality of Service (QoS) goals like scalability, performance, and availability in a distributed computing environment consume a major portion of an application's development time.
- Interoperability across heterogeneous platforms is almost impossible.
- Most distributed computing solutions work well within a LAN, but are not very firewall friendly or accessible over the internet.
- Latency and network failures cause different problems than seen in local processes.

B. WEB SERVICES

In as many publications that describe Web Services there are as many definitions of Web Services. IBM offers this definition:

> A Web service is an interface that describes a collection of operations that are network accessible through standardized XML messaging. Web services fulfill a specific task or a set of tasks. A Web service is described using a standard, formal XML notation, called its service description [WSDL], that provides all of the details necessary to interact with the service, including message formats (that detail the operations, transport protocols, and location.

Our research demonstrates that while one can ascertain there is broad agreement on what Web services might be, there is no single, agreed-upon definition. This ambiguity may be attributed to the opportunities yet to be exploited, as well as the relative immaturity of the technologies. That is to say, we are only at the beginning of conceptualizing and realizing the potential of Web services.

We offer the following definition of a Web service. "A Web service is a distributed computing application that programmatically executes atomic or composite workflows over the network." Web services are essentially about exposing software functionality over the internet. Some known attributes of Web services are less brittle applications, greater flexibility in connecting heterogeneous systems, firewall-friendly, loosely coupled, and greater interoperability. Web services are loosely coupled in the sense that local services retain their unique characteristics while demonstrating the ability to communicate with other systems. Communication between systems may be synchronous, such as the classic client/server request/response model wherein requests must be answered before software processes may continue; or they may be asynchronous wherein messages sent from one entity may go unanswered indefinitely yet processing may continue. Greater flexibility is also manifested in the ability to interconnect applications written in various languages such as Java, C, C++, and Visual Basic and so on. Web services contribute to at least three dimensions of interoperability. For example, Java Web services separate the programming language from the operating system; XML separates data from software; and Web services separate collaborating computer systems, or distributed applications.

Web services are the current phase in the progression of distributed computing, and are based on XML standards and internet protocols. Potential military applications are abundant. The ability to communicate with intra- and inter-Service systems, other DoD systems, as well as systems outside DoD, and to allied systems on the data layer is invaluable. Using open standards like XML and derivative technologies, virtually all systems, legacy and new, will be able to work together as agile, coherent distributed applications.

Web services consist of Application-to-Application (A2A) functionality. Employing platform neutral, open standards Web services interface underlying application components. Web services increase data sharing and A2A interaction without human intervention. Web services are based on the concept of Service Oriented Architecture (SOA).

Web services based on XML standards can be developed as loosely coupled application components using any programming language, any protocol, or any platform (See Figure 23). This greatly eases delivery and consumption of services as anyone using any platform and programming language may access exposed services. The core XML technologies employed for current Web services include Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI). Business-to-Business (B2B) communications have occurred for some time now; however, they were based on proprietary, brittle technologies. Accordingly, scalability of these systems was reduced and very complex.

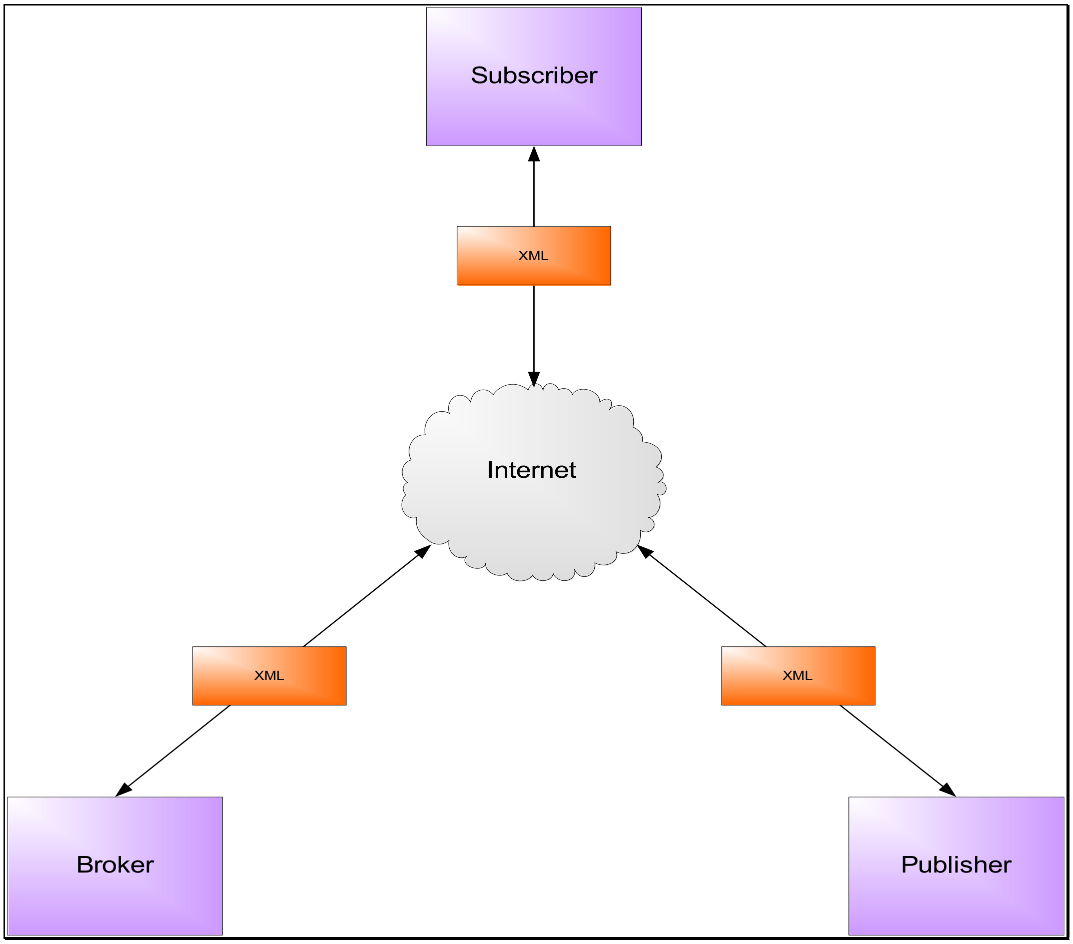


Figure 23. XML for Encoding Distributed Communication (After: Nagappan, 2003, 25).

The basic characteristics of Web services include being based on XML messaging between entities; provision for cross-platform integration; can be built in any language such as Java, C, C++, Perl, Python, C# and Visual Basic. Web services are not intended for handling presentations like HTML. They are based on industry standards like simple, ubiquitous HTTP, and may be easily accessible through firewalls. Web services can be accessed by many types of clients and vary from simple to complex. All major platforms like J2EE, CORBA and Microsoft .NET provide extensive support including open source implementations. Web services can be dynamically located and invoked from public and private registries based on industry standards such as UDDI.

Web services' limitations include the following. They are location specific; that is, a client must be pre-wired with the location of the naming/directory service. They are protocol-dependent; that is, they employ three distinct protocols (SOAP, WSDL, and UDDI). Additionally, some argue Web services are moving quickly toward over-specification and over-complication. [http://judy.jini.org] They are tightly coupled in the sense that a client should be modified or upgraded when the service provider's interface changes. UDDI is a static service broker, and the current Web services model does not allow dynamic deployment of resources and mobile code, only mobile data. (Kumaran, 2001, xv) As well, they do not address the eight fallacies described earlier. [http://judy.jini.org]

1. Basic Operational Model

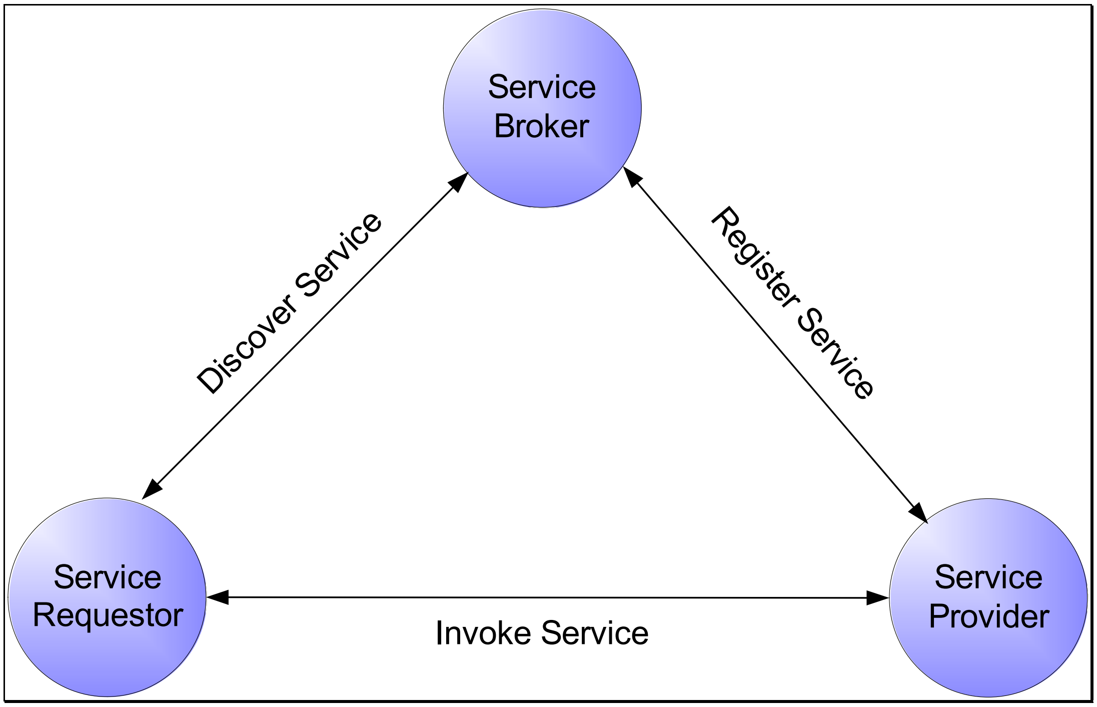


Figure 24. Web Services Operational Model (From: Nagappan, 2003, 27).

As depicted in Figure 24 above, Web services consist of three distinct roles: Service Providers, Service Brokers, and Service Requestors. Service Providers are responsible for developing and deploying Web services. Providers also define and publish services with a Service Broker. Service Brokers, also commonly known as service registries, are responsible for service registration and discovery. Service Brokers list various types, descriptions and locations of the services that assist Service Requestors in finding and subscribing to desired services. Service Requestors are responsible for service consumption. A Service Requestor locates a service using a broker, invokes the service, and executes the service from the provider. Interestingly, the Web services operational model maps closely to the Jini™/CoABS model. Figure 25 diagrams the model used in Jini™/CoABS. A casual observation reveals the operational similarities between Jini™/CoABS and Web services.

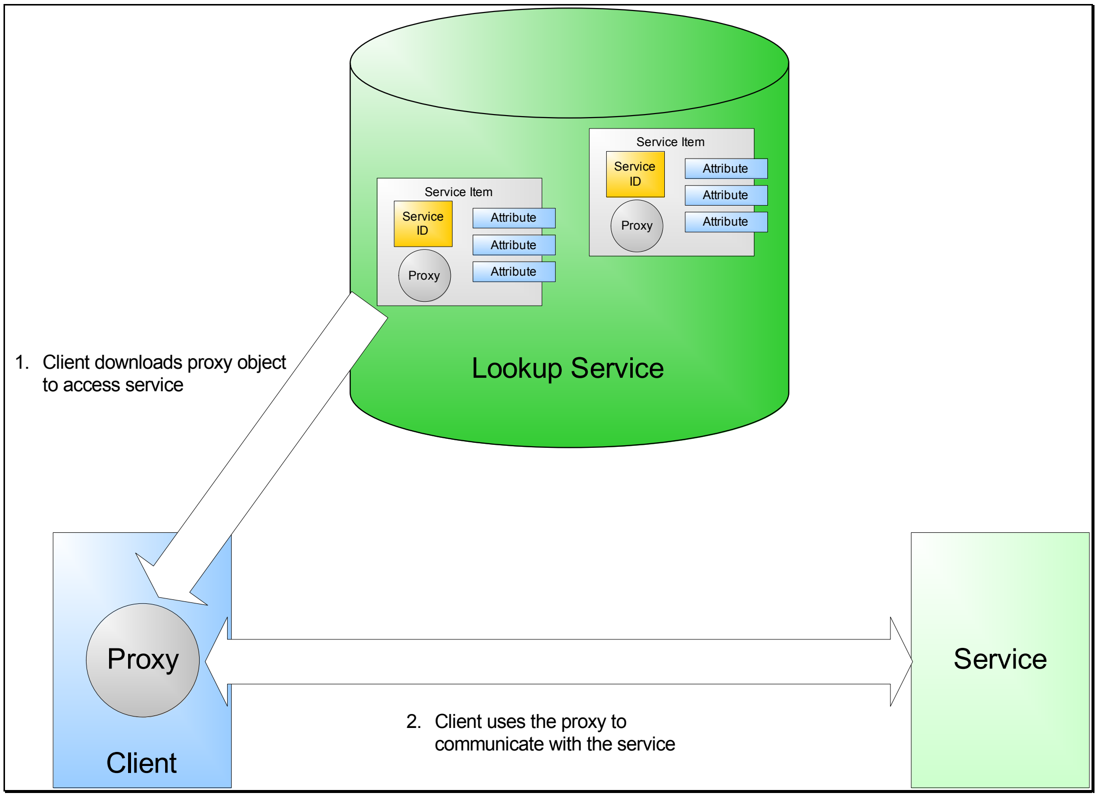


Figure 25. Jini™/CoABS Operational Model (From: Edwards, 1999, 14).

In fact, in at least one test to compare Jini™ and Web services, Jini™ consistently outperformed the Web services (UDDI) in terms of speed, network traffic produced, memory consumption, and scalability.[38] (Schwagli, 2002) Despite these interesting results, we see different uses for Jini™ and (Semantic) Web services. That is, Jini™/CoABS provide a control mechanism and "plumbing" apparatus for autonomous agents (i.e., mobile code); whereas, SWEB services will represent the "small worlds" in

[38] The significant differences in the performance of Jini™ versus Web services may largely be attributed to the fact that Jini™ downloads objects; whereas, Web services are data-stream centric. The transmission of relatively large text across the network requires comparatively more time and bandwidth.

which our agents will interact to return meaningful information. Put another way, Jini™/CoABS provide the infrastructure for Knowledge Acquisition (KA) and SWEB services provide the constructs for the Knowledge Representation (KR) of the service.

The core technology of [current] Web services is XML. XML is manifest primarily in three derivative technologies called SOAP, WSDL, and UDDI. SOAP is a standard for a light weight XML-based messaging protocol. SOAP enables decentralized, distributed communications between two or more participating applications. SOAP is independent of platform, operating system, language or device. SOAP messages may be bound to various internet protocols such as HTTP, SMTP, and FTP. The fact that SOAP is designed to be transmitted over HTTP, unimpeded by firewalls[39] and into places binary protocols such as IIOP, ORPC, and JRMP can't go provides an advantage over other remote procedure protocols. SOAP can be used in synchronous and asynchronous modes of communication. (Moczar, 2002, 373) As SOAP is based on XML, it is extensible. This characteristic accommodates future development and reduces the likelihood of vendor lock-in. SOAP as an XML technology is able to represent abstract data types and so map them between various programming languages. Although there is solid consensus in the industry about the core capabilities of SOAP, there is considerably less agreement on how high-level issues such as security and transaction-management should be addressed. (Graham, 2002, 120) The SOAP specification is available at http://www.w3c.org/TR/SOAP/.

WSDL is an XML derivative technology for describing the network services and their access information. A WSDL service description is an XML document that conforms to the WSDL schema definition. WSDL defines a binding mechanism used to attach a protocol, data format, an abstract message, or a set of endpoints defining the location of services. WSDL is used as the Web service meta-language describing how service providers and consumers communicate with each other. A WSDL document is not a complete service definition, but rather it covers the lower level details of the service – the raw technical information of the service interface. WSDL is an interface definition

[39] Currently, Web services exchange communication using HTTP over the standard web port (port 80). As such, SOAP messages appear to the firewall like any other HTTP request/response; and are therefore allowed through – assuming the firewall allows HTTP service on port 80.

language; that is, as IDL is to CORBA, WSDL is to Web services. (Graham, 2002, 322) WSDL describes the Web service functions offered, location of Web services (e.g. a URL), and how to access the Web service. WSDL-based information is usually stored in a registry, though not required.

Recalling the beginning of this chapter, we stated that one must be able to find the required information sources, understand how to interact with them, make the appropriate requests, and be able to access the results of our requests. A WSDL document supports these goals by describing three fundamental properties of a web service: what a service does – the functions (methods) provided; how a service is accessed – details of the data formats and protocols necessary to access the service's operations; and where the service is located – details of the protocol-specific network address, such as a URL. In the Jini™/CoABS models, the WSDL is similar to the "well-known" interface. Although these tell us where to find the service(s), their existence to begin with must be known or discovered. Both of these methods are aided by a discovery mechanism. Jini™/CoABS services and agents discover then join Lookup Services which store and index service items. Clients discover then query Lookup Services for the needed service or agent. Web services are discovered by finding and querying a registry, namely a UDDI registry. The WSDL specification may be viewed at http://www.w3c.org/TR/WSDL/.

Universal Description, Discovery and Integration defines the standard interfaces and mechanisms for registries intended for publishing and storing descriptions of network services in terms of XML messages. UDDI is similar to the yellow pages, or a business telephone directory. Web service brokers use UDDI as a standard for registering the Web service providers. Web service requestors locate services by communicating with UDDI registries. Web applications interact with the registries using SOAP messages. The scope of registries can be private local area networks all the way to the global, public Internet. The UDDI Working Group includes leading corporations like Sun Microsystems, Microsoft, HP, SAP, and Oracle. More information about UDDI is available at http://www.uddi.org.

Where UDDI implements XML to connect service requestors (clients) and service providers (services), Jini™/CoABS uses the Java programming language to implement description and discovery. However, Jini™/CoABS can implement the mechanisms used by XML Web services as well. The communication protocol used by the service item stored in the Lookup Service is orthogonal to the Lookup Service itself. That is, the Lookup Service is agnostic to the service's (or agent's) "backend" communication protocol(s). Accordingly, Jini™/CoABS could serve as a useful interface to Web services. Both of these methods require prior knowledge of the service; however, this is not entirely a disadvantage. Being required to have foreknowledge of the service and how to interact with it strengthens the legitimacy of the engagement of the service.

2. Known Challenges to Web Services

Three known challenges to successful implementation of Web services include distributed transactions, Quality of Service, and security. If distributed transactions with heterogeneous resources are required, then it should be studied and tested with standard solutions. For mission critical systems, service providers must examine reliability and performance of the service under peak loads and uncertain conditions for high availability. Factors such as load balancing, fail-over, and fault tolerance must be resolved. Military Web service applications must be thoroughly tested prior to deployment to ensure acceptable performance standards. Web services are a promising, but immature technology. Web services are exposed using simple and ubiquitous HTTP-based protocols. Therefore, Web services must be implemented with authentication, integrity and authorization mechanisms using secure SSL-enabling and message encryption technologies. (Nagappan, 2003, 32-33)

Some key benefits of employing Web services include the following:

- Web services provide a simple mechanism for applications to become services that are accessible by anyone, anywhere, and from any platform.
- Web services define service-based application connectivity facilitating intra- and inter-enterprise communication.
- Web services enable dynamic location and invocation of services through service brokers (registries).
- Web services enable collaboration among legacy applications.

Despite the significant advantages and features Web services offer, they can be improved. SWEB services can be implemented to extend the current functionality by enabling agents to execute Web services functionality currently executed by humans.

C. SEMANTIC WEB SERVICES

The Web services model discussed so far is a centralized model. That is, a well known repository (UDDI) is implemented to provide brokerage services for Web service providers (services) and requestors (clients). In this construct service brokers are central to the architecture. As Web services become increasingly sophisticated, brokers will contain varying levels of built-in intelligence, allowing them to learn from transactions; and therefore, they will be empowered to provide better brokerage services. The current WSDL and UDDI technologies are sufficient for design time (static) browsing, and some limited instances of runtime (dynamic) discovery of services. In other words, the human finds the WSDL for a Web service and then develops the client accordingly. However, the next step in description and discovery of Web services must go beyond manual, syntactic discovery and incorporate a layer of intelligence and semantics for true dynamic, human-independent interoperability.

The maturation of existing technologies combined with emerging technologies will lead to further variation in Web services. Accordingly, the ability to effectively operate within this environment will necessitate more agile, more intelligent systems. For example, what protocols will entities use to negotiate transactions? How will entities ensure they are sharing a common language; and the terms used convey a common meaning? How will actors prove their identity? The mechanisms previously discussed help to answer some of these questions; however, more advanced technologies such as the Web Ontology Language (OWL) offer possible solutions to answer more of them.

As previously stated, the current web is designed mostly for human consumption. The SWEB will provide context and meaning in a machine-readable way that will allow machines to understand information stored in the computer. The semantic representation of information will result in much greater levels of interoperability, rich searches, automatic service discovery, and greater task automation. SWEB technologies are closely related to Web services technologies and will eventually converge. (Graham, 2002, 529) The current WSDL description and UDDI taxonomies are intended for human readers to browse. Web service descriptions and registries incorporating ontologies to provide semantics can be machine readable, enabling automatic discovery and invocation of services by software through common terminology and shared meaning. Table 4 from IBM summarizes the differences between SWEB services and current Web services.

Dimension	"Traditional" Web Services	Semantic Web services
Service	Simple	Composed
Requestor	Human	Machine
Provider	Registration	No registration
Broker	Key Player	Facilitator
Service description	Taxonomy	Ontology
Descriptive elements	Closed world	Open world
Data exchange	Syntactic-based	Semantics-based

Table 4. Dimensions of Web Services Compared (IBM).

The notion of description and discovery will persist; however, the manner in which these are achieved will change. Specifically, agents embedded with requisite ontological information will be able to discover and consume SWEB services unassisted by humans. This scenario underscores the intersection, and cooperation, of Jini™/CoABS and SWEB services. Jini™/CoABS will deploy and control agents and societies of agents [KA] that will interact with and consume information accessible via SWEB services [KR].

1. Approaches to SWEB Services

Differing views for how to develop SWEB services exist. Three different approaches to SWEB services described to date include embedding ontological markup within the WSDL; mapping the Web service ontology to the corresponding WSDL; and a different, more expressive methodology termed DAML-S.[40]

[40] DAML is being superseded by a W3C standard called Ontology Web Language (OWL). Accordingly, OWL-S will be used in this document instead of DAML-S.

a. Mapping Approach

One recommended methodology for enabling SWEB services describes mapping an ontology to a Web service WSDL, and is the first case described.[41] In this approach it is argued that current Web services are developed for internal enterprise use; and that most existing implementations are experimental and provide highly focused functionality. Further, for applications where all users are within the same enterprise, manual discovery and coding of clients is the most efficient approach. However, unless the Web service's client knows the exact form and meaning of a service's WSDL in advance, the combination of UDDI and WSDL and coarse-grained operations descriptions is not enough to allow fully automated service discovery and usage.

Currently, semantic confusion occurs which increases the difficulty of accessing and consuming a Web service. Semantic confusion arises from inconsistencies that occur in several ways. These include poorly defined semantics, shared syntax but different semantics, and shared semantics but different syntax. These problems often combine and result in overlapping and conflicting syntax and semantics. Poorly defined semantics manifest as unstated service meanings which must be guessed or interchanged by human actors. Without sufficient semantic descriptions, the service is largely unusable by consumers other than the original developer (provider). Shared syntax but differing semantics can cause confusion between services. This causes problems when a client is coded to use one service, but tries to use another service with different semantics. In this case, as long as the syntax is the same, the call to the service will succeed, but it will return unexpected results. In addition to knowing the correct syntax, the functionality offered by the service must be understood. Even where a WSDL is available, semantics are not expressed precisely in the service interface alone, but must be learned by the (human) developer working on the client. Likewise, two Web services may share similar semantics (functionality) but different syntax. Where this occurs, it is often easy to convert between the two services. However, this is only possible if the meaning of all input and output parameters are understood precisely.

[41] The following discussion is a summary extracted from an article written by Joram Borenstein and Joshua Fox titled "Semantic Discovery for Web Services," published in the Web Services Journal, April 2003 issue.

To solve these problems, a semantic methodology for dynamic discovery and interoperability of Web services must be implemented as part of the overall Semantic Web vision. The key ingredient of the semantic approach is the formal capture of the Web service's interface by reference to an agreed-upon business-oriented vocabulary (semantic model). Further, the vision of the Semantic Web is best implemented with Web services. Our position is that a universal vocabulary is not feasible, even within the military context where structure and formalism is typically greater than commercial systems. We promote the concept of "small worlds" gradually converging based on adaptive techniques, and emerging relations between clients and subscribers. More specifically, we encourage individual units or organizations to build ontologies to satisfy local requirements in agreement with customers. We also recommend lightweight, more easily alterable, modular ontologies that can be readily embedded into software agents for travel across the network. OWL provides facilities for extension and flexibility to capture new concepts and emerging/changing terminologies. For instance, an operational unit may communicate using one set of semantic constructs with its supplier(s), and a different construct with its customers, yet they both may involve the same or similar information. The ontology may be readily adapted to consider the varying terminologies. This is not to suggest that standards be ignored; it simply acknowledges differences will exist, and it emphasizes the ability to adaptively interoperate with varied interests.

For a service to be used, the provider must tell the client what the service "means," and the client must understand what it means and adapt to the service interface. If this requirement is met at a low level and gradually extends as required, semantic interoperability will be achieved on a large scale. This behavior is akin to emergence; wherein, utility and organization manifests over time according to small-scale interests and needs.

The solution proposed requires the service provider use an ontology to model the real-world concepts related to the service's functionality; create a WSDL document for the Web service; map elements of the WSDL messages to the semantic concepts defined in the ontology, saving mappings in RDF; finally, register the semantic model, WSDL, and RDF mappings to the UDDI registry. The role of the service provider is to formally represent the service's semantics in a machine-readable way in a UDDI registry.

In this approach, the semantic model (ontology) represents the meaning of the service's functionality. Where heterogeneous semantic models must be reconciled, they can be integrated by merging them into a larger model that includes synonymous classes and properties. With formal semantic models in place, as long as the service's functionality is encoded in the ontology, syntactic differences can be readily overcome using XSLT transformations to convert between syntaxes. Additionally, in this approach, the ontological model does not yet represent the service itself; however, it does represent the critical concepts necessary to understand the service's functionality.

After creating the ontology, the provider creates the WSDL which defines the service's syntax by specifying the structure of the input and output messages, along with aspects of the service's runtime bindings (e.g. HTTP). The next stage is to express the meaning of the WSDL by mapping operations, along with the schemas of the WSDL input and output messages, to the semantic model. Mappings from the ontology to the WSDL can be expressed in an XML document formatted in RDF. RDF is built on triples expressing a relationship in the form of subject-predicate-object. For example, we might express a mapping as (WSDL element) "serial" (subject) "maps to" (predicate) property "identifier" of class Item (object).

The consumer's role in semantic discovery would be to identify the required functionality; query UDDI, locate the service, and retrieve all semantic and interface definitions; create transformations as needed for the client; and access the Web service using the WSDL document. To identify the required service functionality, the service requestor first discovers the agreed-upon semantic model (OWL ontology) using UDDI and loads it over standard HTTP. The semantic model, even if not standardized, is general enough to describe a wide variety of services for a given industry sector. Where multiple "standard" models exist, they may be linked together through ontological properties and inheritance to form a larger more inclusive model.[42] After the client

[42] Linking of ontologies and/or ontological concepts is difficult and remains a research area.

identifies the relevant ontological concepts in the semantic model, it navigates the mappings that link the semantic model to the WSDL files. The client identifies the semantic values in the model and uses the mappings to find identifiers for the services that provide the needed functionality.

Finally, the client must create transformations to use the service. The client may expect a different syntax than is implemented by the service. However, this problem can be resolved through semantics. In a simpler, but manual approach, a transformation from syntax to syntax is possible using XSLT. However, using semantics the meaning of a service's parameters is clear and transformation code is easy to develop. In fact, in a somewhat more complex application, XSLT transformation may be generated automatically to convert between XML schemas based on the shared semantics. With semantics described and mapped to a WSDL, and transformations in place, the service requestor can invoke the Web service directly just as with any WSDL.

This approach is conceptual and no known implementations exist. It is one possible approach to SWEB services. While we conclude this solution will work, it may not be ideal in that it essentially "bootstraps" a previous technology to a newer one. Given the fact that "traditional" Web services are hardly ubiquitous, we would recommend the military adopt a semantic, ontologically based structure from the ground up. The relative structure that exists within the military domain affords it the opportunity to be a key early adopter of "native" SWEB services. Another approach we encountered was to embed semantic mark inside the WSDL.

b. Semantics Embedded in the WSDL

The next approach adds semantics to the WSDL using DAML+OIL[43] ontologies. It also uses UDDI to store semantic annotations from the WSDL and to search for Web services based on them. One of the goals of semantic enabling of Web services is to automate discovery of Web services. As before, the main inhibitor to this capability is the lack of semantics in the discovery process and the fact that UDDI does not use service description information in the discovery process. Although they assert

[43] DAML+OIL has been subsumed by OWL. All references to DAML+OIL in the original report are replaced by references to OWL, implicitly and explicitly.

that the key to semantic discovery of Web services is having semantics *in the description (WSDL) itself,* and then using semantic algorithms to find the required services, we do not necessarily agree. We view this as another bootstrapping technique that attempts to "bring along" the "traditional" system. Again, Web services are relatively new, so adopting thoroughly semantic structures from the beginning will prove to be more agile in the long term.

An approach for SWEB service discovery is to possess the ability to construct queries using ontological concepts in a domain. This requires mapping concepts in Web service descriptions to ontological concepts. This is conceptually the same as the previous technique; wherein, semantic concepts were mapped to appropriate WSDL elements using the RDF. However, in this approach, WSDL concepts are related to OWL ontologies *inside* the Web service description; that is, in the WSDL document. This technique involves extending the WSDL document to include semantic concepts, thereby enhancing existing industry standards and retaining backward compatibility.

Unlike the previous methodology that suggested no change in the UDDI would be necessary, this approach aims to provide semantic discovery by using UDDI to store semantic information about the Web service. Additionally, an interface is provided to construct queries which use that semantic information.

Semantic annotations added in the WSDL and in UDDI are aimed at improving discovery and composition of Web services. A three phase algorithm for SWEB service discovery which requires the service requestor to enter Web service requirements as templates constructed using ontological concepts is presented. In the first phase, the algorithm matches Web services based on the functionality (operations) they provide. In the second phase, the result set from the first phase is ranked on the basis of input and output concepts of the Web service operations and the input and output concepts in the requestor's template. The optional third phase involves ranking based on the effects.

In summary, this method recommends annotating WSDL constructs with OWL ontological concepts. The designers argue that their approach has the advantage of being an ontologically-based approach that fits better with existing industry standards, without requiring creation of a new infrastructure. The authors indicate that their methodology also contributes to using the extensibility feature of WSDL to add semantics to service descriptions; and to using UDDI data structures to represent grouping of operations with their inputs and outputs.

In our estimation, this approach is useful in that it is a natural progression in the maturation of Web services, and it does not lose backward compatibility already existing in current Web service implementations. However, our research and experience with semantic concepts suggests separating the ontological concepts from other documents to retain modularity, and thus, flexibility. In fact, depending on the scope of the Web service we would recommend the ontology itself be modular and consist of multiple ontology documents. Additionally, as suggested, we recommend the military adopt SWEB service concepts from the onset. WSDL and UDDI stop short of automatic discovery and invocation. They do not readily accommodate composite workflows. Based on our research, we see current Web services performing more singular, atomic processes – with human interaction.

c. Implement DAML-S/OWL-S

The third approach we describe for constructing a SWEB service is derived from a report titled DAML-S: Semantic Markup for Web Services. (Ankolekar, 2003) As part of the DARPA Agent Markup Language (DAML) program, an ontology of services called DAML-S has begun development. Actually, DAML-S will be replaced by OWL-S and will be referenced instead of DAML-S; however, they are quite similar. The aim of this effort is to allow human users and software agents to discover, invoke, compose and monitor Web resources offering particular services and having particular properties. We assert that the OWL-S model will enable these functionalities. The effort described in the report is a collaborative effort between BBN Technologies, Carnegie-Mellon University, Nokia, Stanford University, and SRI International. An important goal of OWL-S is to allow agents to use a Web service automatically. To enable this to occur, the agent requires a machine-readable description of the service. This requirement is common across all SWEB service models.

There are four primary tasks that OWL-S is expected to enable: Automatic Web service discovery; automatic Web service invocation; automatic Web service composition and interoperation; and automatic Web service execution monitoring. Currently, discovery is performed by a human who might use a search engine to find candidate services, select the most appropriate service, develop a client, and finally consume the service. OWL-S is expected to automatically discover the service through computer-readable semantic markup, and a service registry or ontology-enhanced search engine; or a service can be exposed in OWL-S with service registry, so that requestors can find it during the service search phase. Having automatically discovered a Web service, a software agent should be able to automatically invoke the Web service. Instead of a human "completing a form" and submitting the required data to the service provider, the agent could be delegated the task of executing the necessary functions of the Web service. In this way the human becomes further removed from the process. Provided with the overarching goals, a given task may be automatically performed. To accomplish this, the OWL-S model would encode the information necessary to select and compose Web services at the service site. To allow service consumers the ability to monitor the status of their service request, OWL-S will provide automatic Web service execution monitoring.

In the OWL-S paradigm, the class *Service* is the top level class in a taxonomy of services, and its properties are normally associated with all kinds of Web services. An ontology of services is specified to describe three essential types of knowledge about a service. An ontology by its very nature is intended to describe the meaning of something; that is, it is a specification of a conceptualization. To that end, an ontological framework should be devised to answer certain questions about a given conceptual domain. In the context of Web services the OWL-S ontology for services seeks to answer three questions:

What does the service require of the user(s), or other agents, and provide for them? OWL-S answers this question in the service "profile;"[44] so the class *Service presents a ServiceProfile*.

[44] "Profile" is also known as a service capability advertisement (K. Sycara, 1999).

How does it work? OWL-S gives this answer as the service "model;" so class *Service* is *describedBy* a *ServiceModel.*

How is the service used? OWL-S answers this question as "grounding;" so the class *Service supports* a *ServiceGrounding.*

The properties *presents, describedBy,* and *supports* are properties of the class *Service.* The classes *ServiceProfile, ServiceModel,* and *ServiceGrounding* are the respective ranges of those properties. The service profile tells the agent seeking a service what the service does so the agent can determine whether or not the service meets requirements. The service model tells how the service works, or what happens when the service is executed. Finally, the service grounding specifies the details of how an agent can access a service; for example, the grounding specifies the service's communications protocol. The *ServiceProfile* provides the information needed for an agent to discover a service; and together, the *ServiceModel* and *ServiceGrounding* service objects provide information for the agent to make use of a service. Figure 26 depicts the top level service ontology.

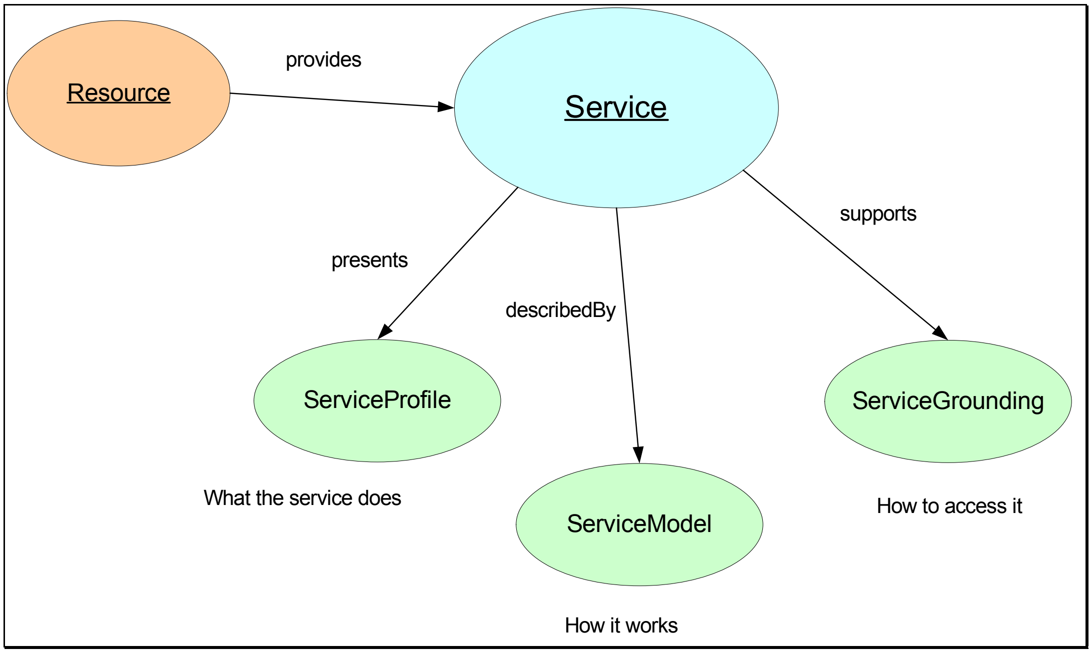


Figure 26. Top Level of Service Ontology[45].

[45] [http://www.daml.org/services/daml-s/0.9/daml-s.html]

In our estimation, the OWL-S model essentially separates the Web service functions described in the WSDL document into the *ServiceProfile* and *ServiceGrounding* ontological classes. The *ServiceModel,* to a lesser extent, represents a portion of the WSDL as well. The OWL-S report acknowledges that the *ServiceProfile* and *ServiceGrounding* classes present the industry standards functionality (i.e., the WSDL). However, we agree that the ontological markup like that demonstrated in OWL-S is essential to enable richer queries and further automation. The OWL-S process would certainly enhance a Web service's functionalities

We believe ontological markup that enables machine-readable representation of information is essential to Web services and KR. Of the three approaches discussed in the research, the modular approach of mapping a service ontology to a WSDL causes the least upheaval for existing Web services. We conclude that modularity supports increased flexibility for future development efforts. The second approach, which embeds the ontological markup inside the WSDL, we argue decreases flexibility and increases the brittleness of the service. Additionally, it asserts that the Web service broker component needs no changes. Significantly, we contend that the UDDI should include semantic markup as well. The OWL-S concept appears to provide a potentially useful solution, and is the recommended approach for new Web services development. In summary, an approach that provides semantics, is modular, and extends or enhances current standards seems most beneficial.

D. SUMMARY

So what does all this mean? How does the military use these technologies effectively, and for what purposes? If we accept the premise that the paradigms described represent the progression of distributed computing, we can make some significant observations and infer some utility that may be derived. One observation is that the most "advanced" model described in the list, CoABS, still heavily uses the client/server model by sending executable code between Lookup Services and requesting clients using simple HTTP servers. In fact, arguably, all current distributed computing models to date are client/server models (Booch, 2001, 36)

Another observation is that all complex workflows can be reduced to discrete atomic steps. SWEB services can be implemented to describe and execute complex organizational workflows with little or no human involvement. Interchanges between military commands can be accomplished automatically using SWEB services and autonomous agents. This can potentially increase the military commanders' decision rates by getting the right information at the right place sooner and more reliably.

We also conclude that Jini™/CoABS are forms of a web service. In fact, we can describe Jini™/CoABS as a Service Oriented Architecture. The Lookup service is somewhat synonymous to the UDDI registry in that it contains a list of all available services (and agents). It allows potential clients to lookup and access services, and allows agents to send messages between each other. For instance, a service client sends a query to the Lookup service; the lookup service finds the service, or services, that satisfy the query request and returns the proxy to the requesting client. Similarly, in the web services model the service requestor uses the UDDI to find a service; then the service requestor (client) consumes the service from the provider.

But Jini™/CoABS give us more. It not only provides the UDDI functionality but it provides automatic service discovery and joining. It also provides a leasing facility which purges failed services automatically and provides a self-healing characteristic. Within Jini™/CoABS we not only enjoy service discovery and invocation, we also realize messaging and agent functionality (i.e., mobile code). The CoABS Grid provides a well-defined environment within which the military community may interact. Services may enter and exit the system automatically, without disruption to the larger system.

Additionally, SWEB services can be dynamically listed, found and consumed from within CoABS. Although RMI is the default transport mechanism for executable code, and performs well, it is certainly not required. SOAP over HTTP as used in the XML Web services paradigm may also be used in Jini™/CoABS.

In addition to providing a container for sharing services, CoABS implements a message queue system. This is similar to the message oriented middleware model and it allows services to be extended to agents. The agents are essentially services that implement a message queue. So we can infer that CoABS is an instance of a Service Oriented Architecture combined with an agent system. Semantic messaging will allow more sophisticated agent communication to occur – further enhancing their utility.

Jini™/CoABS are limited in automatic discovery however. Specifically, a service item is described by a unique identifier that distinguishes it among services, a well-known interface that tells a client what it does how to interact with it, and a collection of attributes that describes the service. In some ways, this concept may be mapped to the WSDL in that it contains identifying information about the service and how to interact with it. Clients may use any combination of these characteristics to search for services or agents that meet specified criteria. However, this implies that the client has prior knowledge of the service or agent. That is, it has the service's or agent's well-known interface. In the case of agents, the well-known interface is used system wide. Still, the specific agent required must be isolated among all other agents present. Services also provide a well-known interface; however, the specific services provided are unique and varied. The crux of Jini™/CoABS is prior knowledge is required; otherwise, it appears to be an exceptional solution to store, transport and manage services and agents.

With a solution for the deployment and control of agents available to us, along with a semantic method for representing information, we have realized the ability to allow mobile code access to information represented in the computer; that is, we have achieved knowledge acquisition and knowledge representation. We have smartly connected small worlds of information. Having achieved access to machine readable information, the next chapter discusses concepts of software agent theory and presents two example implementations of agents.

V. AGENTS

A. BACKGROUND

This chapter discusses the notion of software agents, their relationships to each other and to humans, and considerations for their employment in military operations. Additionally, this chapter discusses a sampling of the agents we developed in support of an agent-based prototype application called "ArchAngel." The purpose of this chapter is two-fold. That is, it provides an abstract conceptual underpinning for agents, and follows with concrete illustrations.

Agents in the context of the Semantic Web (SWEB), or any networked architecture, will prove indispensable. Software processes described as agents are already in limited operation. We argue that many of these so-called agent implementations are loose interpretations of "autonomous agents;" and in fact, are possibly described as agents for marketing purposes. Nonetheless, we use a fairly flexible interpretation of agents in our work, aiming to capitalize on autonomy, messaging and mobility – three of the many essential aspects of agents.

To some, the idea of autonomous agents acting on our behalf may seem ridiculous, futuristic, or even dangerous. Others may view the idea of agents on the network as revolutionary. We settle somewhere in the middle. That is, we believe the wide-scale application of agents will eventually happen as a product of necessity. We also assert that agents will shoulder the burden of many of the mundane tasks humans currently perform. On the other hand, we also believe the ubiquitous presence of agents will emerge subtly.

B. AGENTS DEFINED

1. What Are Agents?

There is no universal agreement on what a software agent is. In fact, there is controversy among the experts concerning the exact definition of an agent. (Tanenbaum, 2002, 173) For the purposes of this paper, it suffices to define a software agent as a software process capable of reacting to percepts (input), and initiating changes in its environment (output), possibly in collaboration with human decision makers or other software agents. This is a broad definition, allowing different types of software processes to be described as agents. Figure 27 shows the most basic function of an agent: to receive percepts from its environment and output actions to its environment.

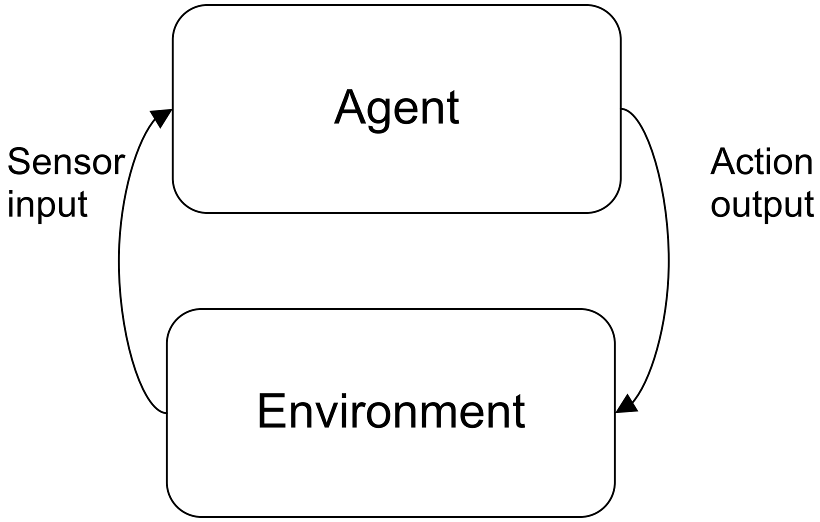


Figure 27. An Agent Takes Sensory Input from Its Environment and Outputs Actions That Affect It. This Interaction Is Typically Ongoing, and Non-Terminating. (From: Weiss, 2001, 29).

There is no widely-accepted, standard classification guide for agents as yet. However, for the purposes of our discussion, we will enumerate some of the basic types of agents. In most instances agents should be autonomous[46], and they should also be able to cooperate with other agents. The combination of autonomy and cooperation leads to a class of agents called "collaborative agents." Collaborative agents seek to achieve common goals through cooperation. For example, collaborative agents might be used to schedule a meeting between human actors. Another type of agent commonly described is a mobile agent. A mobile agent is able to move around to different machines. Although not mandatory, many agents require mobility. Indeed, their autonomous and interactive nature normally dictates agents possess mobility. The class of mobile agents is not mutually exclusive from other classes of agents. In fact, other classes of agents may be mobile. From a functional perspective, other types, or classes of agents emerge. One type of generally accepted class is interface agents. This class of agents typically assists

[46] By "autonomous" we mean the agent is "authorized" to make decisions on our behalf.

an end user in the use of one or more applications. A distinguishing characteristic of interface agents is the ability to learn. As the interaction between the user and the agent increases the agent is able to provide improved support. Closely related to interface agents are information agents. Their main function is to manage information from many different sources. Information agents may also be referred to as “keeper agents.” (Weiss, 2001, 428)

Significant attributes of agents include reactivity, pro-activity, and adaptability. Additionally, communication, self-healing and continuity, or long-lived-ness, are key traits. Reactivity implies that the agent effectively and efficiently takes a specified action in response to inputted percepts. An added degree of sophistication is for an agent to possess the ability to initiate actions that cause changes in domain state. The ability to learn, or adapt, is another defining trait of agents. This was discussed with the interface agent class. Communication is essential for collaboration and any degree of sophistication and to be sure is common to all agents. In addition to all these traits, which not all agents must possess, continuity is important for many applications of agents. The ability to operate over long periods of time is central to many agent activities, as we intend to delegate some tasks to them simply because of endurance. Self-healing agents are vital to many applications. When systems crash or network connections are lost, many agents must be able to be automatically reconstituted.

As we described earlier, Jini is designed to handle precisely the ability of agents to automatically recover from system and/or network failures. Jini/CoABS also directly enable many of the other attributes such as continuity, mobility, communication, and interface. All these traits may be directly or indirectly implemented using Jini/CoABS. Table 5 summarizes important distinguishing properties/classes of agents.

Property	Common to all Agents?	Description
Adaptive	No	Capable of learning
Autonomous	Yes	Can act on its own
Collaborative	No	Autonomous and communicative
Communicative	Yes	Can exchange information with users and other agents
Continuous	No	Has a relatively long life span
Information	No	Manage information; "keepers"
Interface	No	Interaction with human; learning
Mobile	No	Can migrate from one site to another
Proactive	Yes	Initiates actions that effect its environment
Reactive	Yes	Responds timely to perceived changes in its environment
Self-healing	No	Capable of self-reconstitution

Table 5. Properties of Agents (From: Tanenbaum, 2002, 175).

Any control system can be viewed as an agent. An oft-cited example is a thermostat that automatically changes its output from one of two states when it senses changes in its environment's temperature. This behavior is ongoing and automatic. Of course, more sophisticated control systems, with more elaborate decision systems exist. Examples include autonomous space probes, fly-by-wire aircraft, nuclear reactor control systems, etc. (Weiss, 2001, 31) Most software daemons (disk execution and monitor), which monitor a software environment and perform actions to modify it, can be viewed as agents. In section E we detail some of the agents developed during our research, but now we discuss the notion of "intelligent agents."

C. INTELLIGENT AGENTS

Intelligent agents may be described as agents capable of flexible, autonomous actions executed to meet their design objectives. Flexibility in this context refers to reactivity, pro-activeness, and social ability. Reactivity means agents are able to perceive their environment and respond to changes in a timely manner to achieve design goals.

Pro-activeness refers to the goal-seeking behavior of an agent that takes the initiative to satisfy its design criteria. Social ability means agents collaborate with other agents to achieve goals.

1. Agent Decision-Making

Inasmuch as humans rely on each other to make timely, effective decisions, we also require a degree of confidence to rely upon agents to make coherent, rational decisions. Autonomous, mobile, reactive agents will assist the human decision maker in a variety of ways. For example, they will help by increasing the information analysis rate, thereby mitigating analysis backlog. The reduction in analysis backlog will indirectly lead to increased knowledge yield and then more actionable intelligence (new knowledge). The end result: a greater effective decision rate, and reduced – not eliminated – uncertainty in warfare.

2. Decision Trees

Human decisions are made based partly on experience (i.e., memory). One may recall facts about a given situation and assign probabilities to certain outcomes occurring. Formal decision trees may be readily developed to capture and describe expected outcomes for a given scenario and its associated options. Similarly, an agent may be fortified with memory in the form of a database or knowledge base. As Figure 28 suggests, an agent provided with state memory will be able to consider past states of the environment to decide its next actions. A mechanism such as Bayesian belief principles[47] may be combined with the agent's memory, and its design goals to enable the agent to make decisions based on perceived sensory inputs. In this highly mechanistic structure, agents would lack the inference capabilities of humans; however, the incorporation of memory (i.e., database) enables agents to effectively learn over time and experience, much like humans. Simple inferences would be achievable relatively early in the life of an agent. As the agent developed a deeper memory, it would be able to provide more reliable and complex inferences for a given problem domain.

[47] Bayesian belief systems are but one example; classification and subsumption are two others.

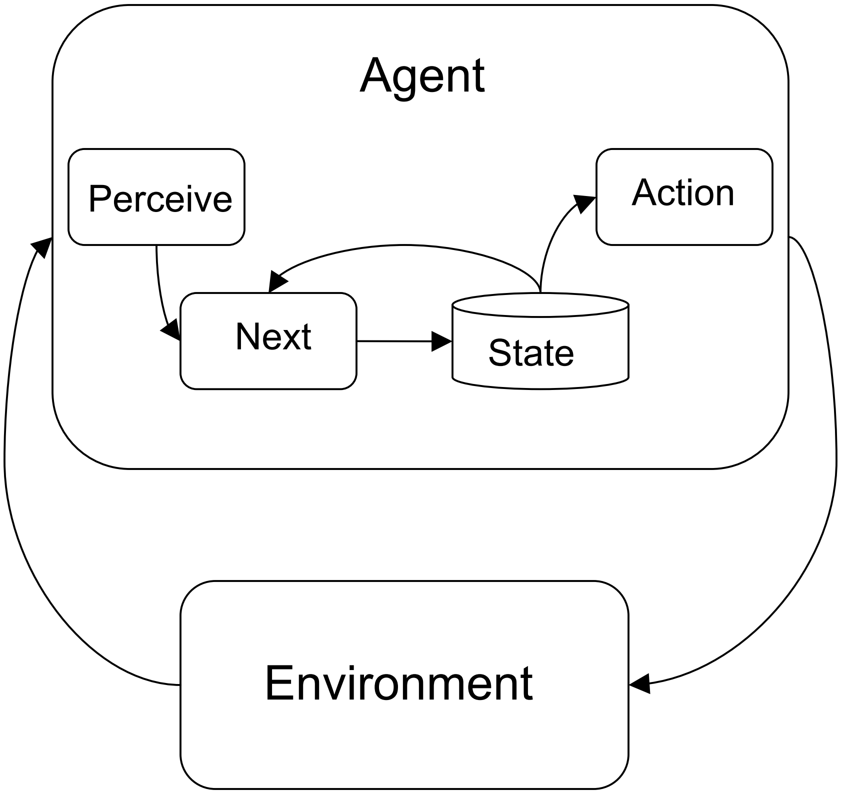


Figure 28. An Agent Fortified with Memory (From: Weiss, 2001, 41).

The incorporation of state within an agent should not be confused with the notion of an agent interacting with a knowledge base. When we refer to state “within an agent” we mean to indicate the agent itself possesses memory. We seek to enable the agent to acquire, over time, increasingly sophisticated inference capabilities. This characteristic enables the agent to more intelligibly interact with a knowledge base to produce richer inferences as previously mentioned.

3. Agents in Action

The implementation of agents supporting the military commander would include, but not necessarily be limited to, the ability to provide situation-dependent alerts based on design criteria. For example, the proximity of an enemy unit relative to own forces may trigger an alert. While this sort of capability is not unique to current operations, it is enhanced by the ability of the agent to also provide warnings. Building on the previous example, an alert could include one or more warnings. The agent could advise the commander that the enemy unit is not only a certain proximity to own forces; it could provide a warning describing its capabilities, possibly based on the agent's organic memory of this particular unit, or through the agent's interaction with a knowledge base - or some other means. This characteristic could be further improved by the provision of statements of implications and recommendations for action. The agent might "know," based on its internal memory and built-in intelligence, or through consultation with a knowledge base, that a given unit may be superior in force to our own. Consequently, it might recommend evasive action and provide potential effects for engagement instead.

4. Adoption Inhibitors

Inertia will lead to skepticism and reluctance to incorporate software agents. The willingness of a military commander to rely on software agents who propose courses of action in real time may be marginal – at least initially. An implementation that "overlay" agents atop extant situational awareness systems will help the agents establish trust with the human decision makers. The implementation should not materially alter current systems, and, in fact, could be optionally available as a service. Just as new personnel must earn trust when joining a group or organization, so to must a software agent; it is a universal norm. Over time and trials, feedback from the agent will be gradually factored into operations, as long as the agent demonstrates reliable outputs. Agents should be tested thoroughly, before and after deployment, to ensure they are rational and are properly designed. Additionally, agents must be (re-)configurable by operators who only know the mission requirements and not the technical details of the agents' inner workings. The requirement to minimize the OODA loop and to maintain information dominance will necessitate large-scale integration of software agents into various systems, including Command and Control systems. Software agents operating inside semantic constructs will necessarily pervade these systems, enhancing current decision-making processes.

Notably, at least one possible unintended consequence that may result, however, is that the increased decision rate may simply lead to increased available information as we learn more and more through our actions. This situation essentially places us where we currently are: striving to manage too much data. The difference would be that we are confronting too much information. However, this is speculative, and it does not obviate the need to increase our effective decision rate today.

D. AGENT EXAMPLES – CONCRETIZING THE CONCEPTS

1. ArchAngel Agent Based System Prototype

ArchAngel is a prototype SWEB system that addresses the very basic starting point of how software agents can be used to enhance the ability of the war fighter. The objective of ArchAngel is to examine the use of SWEB technologies and agents interacting inside a sensor grid[48]. Ultimately, the agents using rules and ontologies will interpret instance information represented in the computer (semantics) and provide alerts, warnings, recommendation actions, and statements of implication to the human decision maker(s). It is worth noting that, to date, the technologies used to determine how software agents can enhance mission effectiveness are primarily open source, based on W3C Recommendations. One of the questions ArchAngel seeks to address is whether or not the SWEB is a necessary and sufficient technology to military operations, specifically the Expeditionary Pervasive Sensing (EPS)[49] program.

The functional intent of the ArchAngel project is to retrieve valuable information from heterogeneous[50] data sources and assemble it into a master operational context document for storage in a rudimentary knowledge base (KB). This document will be written in the OWL and will be used to conduct reasoning operations in an effort to increase programmatically intelligence yield and realize new knowledge.

[48] The notion of a grid is similar to the power grid. That is, an interconnected network of sensors in concert. Failures in one part of the grid do not necessarily affect other portions of the grid.

[49] EPS is an Office of Naval Research (ONR) sponsored program run by the Naval Warfare Development Command (NWDC) designed to investigate agent-based technologies to support a future battlespace with a proliferation of ISR sensor systems. (See[http://www.nwdc.navy.mil/Concepts/EPS.asp]).

[50] By "heterogeneous" we mean different data models, such as relation, XML, object, etc.

Presently the ArchAngel project includes two primary agent "teams." They are informally called the "REPEAT"[51] and "Message" agents. Each team represents one of many agent sets (societies) contributing to the larger goals of the system. The Message agents were designed to support Personnel Recovery (PR) message flow functions; whereas, the REPEAT agents were designed to act upon (simulated) Navy Over-The-Horizon (OTH) Gold messages. Both agent teams implement a design pattern we refer to as the "Agent Triad." (See Figure 29).

2. Agent Triad

The concept underlying the triad design pattern is that one agent acts as a "monitor" or "watch/listen" agent; a second agent acts a "broker" or "controller," and the third acts as a "handler" agent. It is not mandatory that one agent per function exist; it is possible, even likely, that a number of agents may be employed to accomplish the overall goals. However, the triune combination appears well suited for a variety of application instances. In fact, as we will observe, the Message agent team implements several agents to accomplish the handler function; whereas, the REPEAT agent team consists of only three agents. Generically, the monitor agent will watch or listen to a specified source of data. The source may be incoming messages or state changes (e.g., changes in database), or a host of other sources. The broker agent, based on messages from the listener agent, will make a determination on a course of action based on design criteria, and will act accordingly. The actions may include, but are not limited to, taking action itself, notifying the handler to perform some action, or possibly directing the listener to execute some activity. The REPEAT agents and the Message agents each follow a similar intra-agent messaging pattern, and will be detailed shortly. But first, it is necessary to discuss other characteristics that both agent teams share.

[51] "REPEAT" comes from a SPAWAR software application that is capable of transmitting simulated test messages over serial COM ports and TCP/IP connections. We used the latter. The REPEAT software is produced by SPAWAR. See [https://repeat.spawar.navy.mil] for more information.

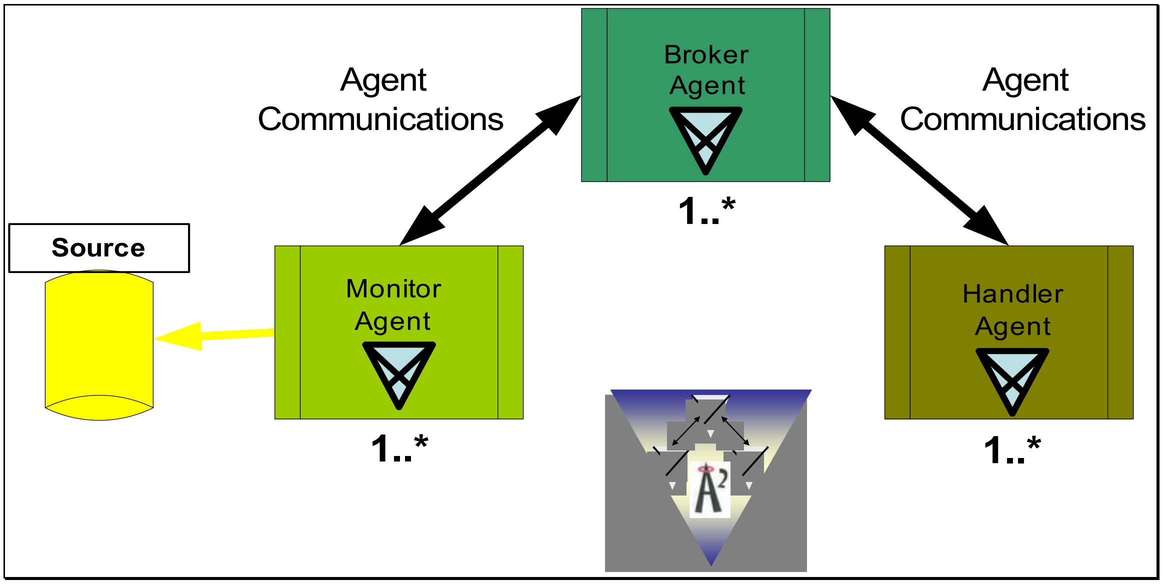


Figure 29. Agent Triad Design Pattern.

The agents were developed using the CoABS API. Additionally, they operate and function within the CoABS Grid software. As described in the Distributed Computing chapter, CoABS is an extension of Sun Microsystems Jini™ technology. Some of the features of Jini/CoABS that made it attractive for use in ArchAngel included automatic lookup and discovery of services and agents, mobile code, the self-healing nature, and a messaging function that allowed the agents to communicate in a loosely coupled way. These features combined to enable a robust distributed computing application; one we feel is central to a SWEB application. The REPEAT agents and Message agents were built on Java technology, implemented open source XML technologies, and performed exceptionally well. We now discuss two of the agent teams in turn.

3. REPEAT Agents

a. Overview

The REPEAT agents are a building block of ArchAngel. The idea behind the REPEAT agents is to simulate Global Command and Control System – Maritime (GCCS-M) messages to reflect near-real-time status of the maritime battlespace. The data source used for the REPEAT agents is a software application called - predictably - REPEAT capable simulating the transmission of various text messages. The messages used for our simulation are customized, fictitious Over-the-Horizon (Gold) or OTH Gold messages. These messages are contact reports adhering to the OTH Gold specification[52], and are human-readable, machine-parse-able messages. These messages are transformed to XML, then stored and manipulated in the Xindice XML database.

We used the Agent Triad design pattern to intercept and manipulate the incoming messages. This triad consists of just three different software agents. They include an agent that listens for messages (RepeatListener), an agent that serves as a broker of the messages (RepeatBroker) and a message handler agent (RepeatHandler).

The overall function of the RepeatListener in this application is to intercept the text messages, transform them to XML, store them in an XML database, and finally, send a notification message to the RepeatBroker.

In addition to the agent classes, there is an administrative package of classes that facilitates deleting files from the various database collections used in this project. It is called “RemoveResources,” and it makes a connection to the XML database and iterates through each of the defined collections deleting all resources (XML documents) in each collection. This helper class makes collection management much easier by automatically deleting resources between simulations.

Another class in the admin package is the “MyXpath” class. This class will retrieve the unit names from the “Unidentified” collection. This class could serve as part of an agent as ArchAngel is further developed. One potential use is to peer into the unknown unit files and make decisions about those units, and/or notify the human decision makers. We now walk through a thread of operation for the REPEAT agent team.

b. Functional Flow

First, the OTH Gold text messages were constructed for our scenario. We chose to create two XML files that contained all the necessary data elements for the entire scenario. Each file contained fictitious enemy units, some of which were

[52] The document we referenced is titled “Operational Specification for “Over-The-horizon Targeting Gold Revision D”, and was published under direction of CNO N62 by Navy Center for Tactical Systems Interoperability, 1 September 2000.

stationary (static) while others were mobile (movers). One file included all the units, static and movers, and was used to initiate the scenario. The other XML file contained only the movers, and their various locations to be plotted over a five day period. Once completed, the scenario XML files were transformed into numerous OTH Gold text messages in accordance with the OTH Gold specification mentioned earlier. With the text files built, our data source was ready to be used by the REPEAT, specifically the RepeatListener.

With all agents operating inside the Grid, the REPEAT software was configured to begin transmitting the text messages over a TCP/IP connection (on port 2021). Once received, the RepeatListener would capture the text transmission and transform it to an XML representation. Then, the OTH (XML) file was temporarily stored in a Xindice collection called "TempStorage." Then a message was sent to the RepeatBroker agent for further action.

The RepeatBroker, upon receipt of a message from the RepeatListener, retrieved the stored XML document, and then checked the latitude and longitude values to see if the unit was currently, or was previously, in the Area of Operations (AO). If the unit was currently in the AO, or was previously in the AO, the RepeatBroker sent a notification message to the RepeatHandler. If the unit was not presently in the AO, and was not previously in the AO, the OTH (XML) message was deleted by the RepeatBroker and no further action was executed. The determination whether or not the unit was in the AO was made by using regular expressions based on the known coordinates of the AO. The test for whether or not the unit had been in the AO previously was made by comparing the unit's name to the unit collections. The intent was not as much to make sophisticated operational or strategic decisions, but rather to exercise agent characteristics such as communication and reactivity.

If the unit was currently in the AO or had been in the AO, the RepeatHandler received a message from the RepeatBroker. It then determined in what database collection the XML document should be stored. We developed Xindice collection hierarchically structured based on the unit names. The agent inspected the unit name and used it to determine where to store the XML document. The unit names were mapped to collection locations (Xindice URLs) using a hash map. If there was no known collection to store the XML document, the RepeatHandler stored it in a collection designated "Unidentified."

The flow of information and messages between the agents is depicted graphically below (See Figure 30). A couple of items to note include the following. First, the diagram illustrates a computer with the text "Tomcat server" coming into it. Apache Tomcat[53] is an open source application server that we used to host an application called Xincon. Xincon is an (open source) web-based application that allows one to perform basic database management activities on Xindice, such as adding and deleting documents and collections. The output side of the computer in the figure is a screen capture of Xincon depicting the collections used in the REPEAT agent team. Above the screen capture we show sample OTH XML files that may be used to provide a 3D presentation of the scenario in the browser. As we will observe in the section describing the Message agents, we are able to display the important message information items in a browser; however, this functionality is not yet completed for the REPEAT agents.

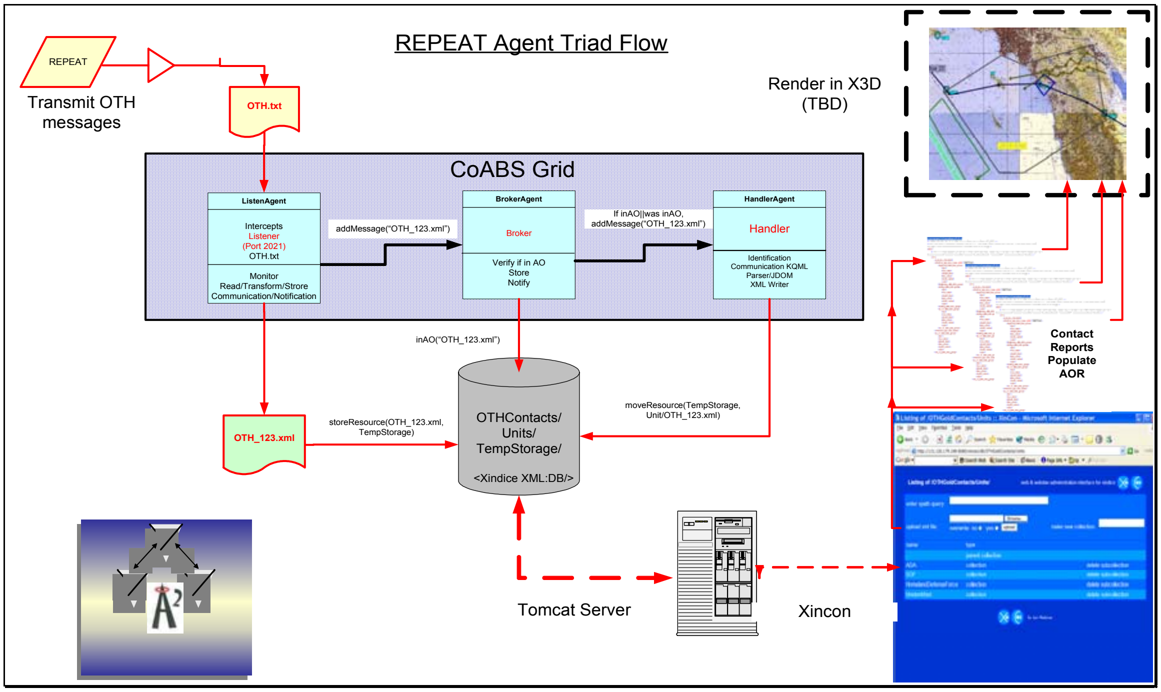


Figure 30. REPEAT Agents Functional Flow.

[53] Tomcat information is available at [http://www.apache.org/jakarta].

c. Control Flow

This section provides a more technical survey of the control flow inside and among the REPEAT agents. We step through a System Sequence Diagram (SSD) that captures the communications among agents and significant method activity. As we observe from Figure 31, the process is begun by starting and registering the various REPEAT agents in the grid.

First, we start the HandlerAgent providing an agent name; and then it is registered in the grid. Next, the BrokerAgent is started and then registered with its name. Likewise, we register the ListenerAgent. With the REPEAT agents ready, the REPEAT data source begins transmitting OTH Gold text messages on port 2021. There may be any number of messages transmitted, and the REPEAT source will continue until it has emptied its "queue," or until it is manually stopped.

With the ListenerAgent bound to port 2021, each text message is intercepted and transformed to an XML representation. The resulting XML document is stored in the TempStorage collection inside Xindice by the ListenerAgent. After the XML document is stored, the ListenerAgent sends a message to the BrokerAgent passing the saved XML file name, and the location it was stored.

The BrokerAgent gets the XML document (resource) from the TempStorage collection, and then checks the unit's location described in the XML document to see if it is in the Area of Operations (AO), or *was* in the AO. If the result is "false" the XML document is deleted from the database. If the result is "true" the BrokerAgent sends a message to the HandlerAgent, providing the name of the resource.

The HandlerAgent retrieves the resource from the TempStorage collection and then determines whether or not the unit described in the resource is already contained in a collection inside the ArchAngel collections of enemy units. If it is then the resource (XML document) is stored in its unit's collection (and the original is removed from the TempStorage collection). If the described unit is not contained in a collection, the resource is moved to the "Unidentified" collection. This completes a cycle for the REPEAT agents; however, this process continues for as long as the REPEAT data source transmits OTH messages.

The REPEAT agents are not intended to be a sophisticated agent society; however, they do work cooperatively within CoABS to complete an objective. They also demonstrate communication and reactivity. They perceive inputs and provide appropriate outputs. The REPEAT agents could be improved by implementing a formal Agent communication Language (ACL), and interacting with a knowledge base consisting of formal rules, ontologies, and marked up instance data. Additionally, the logic embedded in the agents could be enhanced to cause behavior suitable to the domain. Finally, the application could be made less brittle by implementing more generic algorithms for determining in what collection a unit belongs.

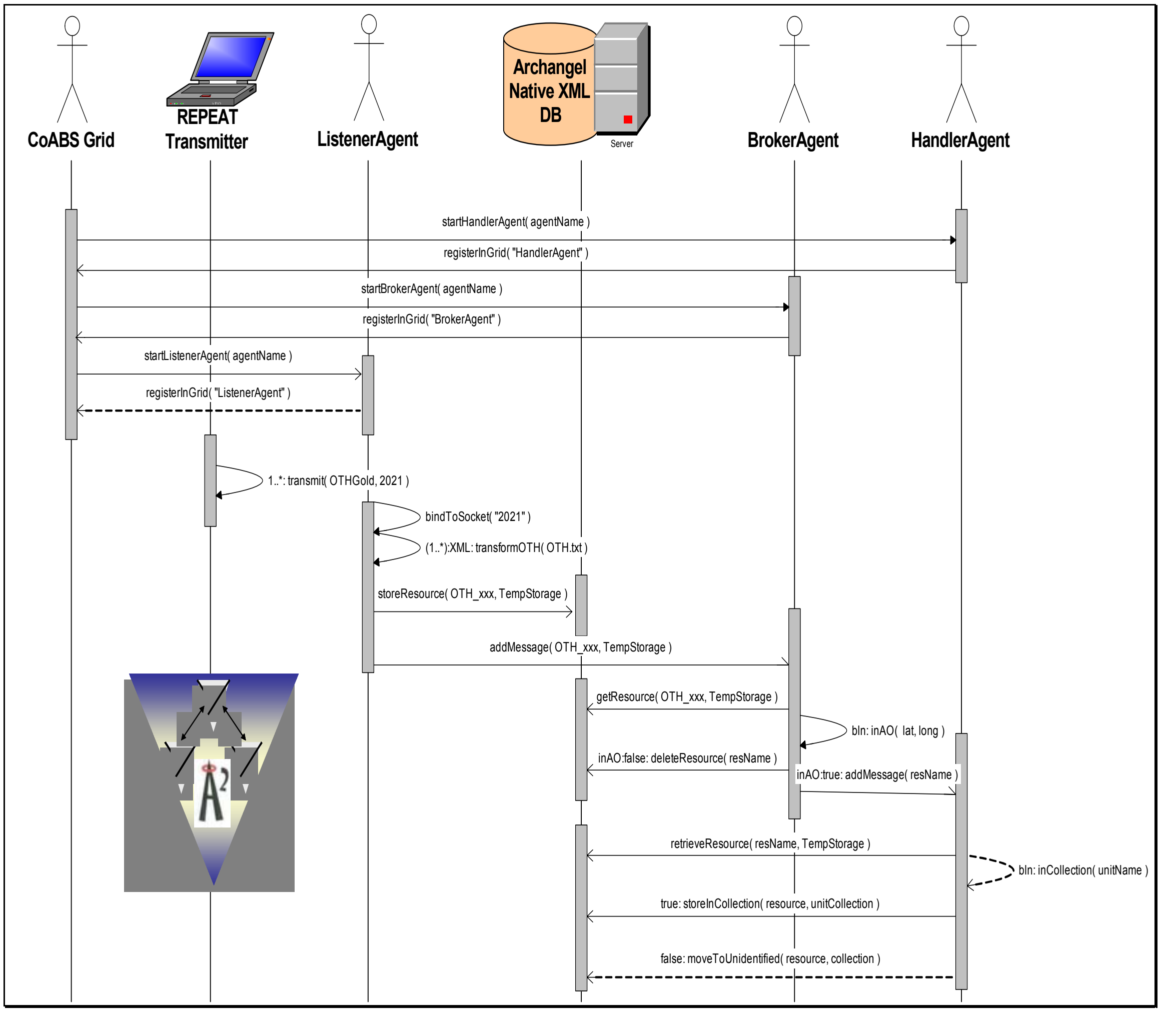

CoABS Grid
REPEAT
Transmitter
ListenerAgent
Archangel
Native XML
DB
Server
BrokerAgent
HandlerAgent
startHandlerAgent(agentName)
registerInGrid("HandlerAgent")
startBrokerAgent(agentName)
registerInGrid("BrokerAgent")
startListenerAgent(agentName)
registerInGrid("ListenerAgent")
1..*: transmit(OTHGold, 2021)
bindToSocket("2021")
(1..*):XML: transformOTH(OTH.txt)
storeResource(OTH_xxx, TempStorage)
addMessage(OTH_xxx, TempStorage)
getResource(OTH_xxx, TempStorage)
bln: inAO(lat, long)
inAO:false: deleteResource(resName)
inAO:true: addMessage(resName)
retrieveResource(resName, TempStorage)
bln: inCollection(unitName)
true: storeInCollection(resource, unitCollection)
false: moveToUnidentified(resource, collection)
A2

Figure 31. REPEAT Agents System Sequence Diagram.

d. Example Code[54]

Below is a code snippet from one of the REPEAT Agent classes (RepeatListener). It first declares this class' package name (i.e., navy.nps.archangel.repeat.listener). Next it imports the core Jini classes it requires. All three of the Jini classes imported will be used in combination to retrieve the RepeatBroker's proxy from the Lookup service. In this case, the Jini Entry class is used to contain the name of the RepeatBroker. The ServiceTemplate class will enable us to build what is essentially a query that will return a ServiceItem. In this case the returned service item is the proxy for the RepeatBroker.

The next set of imported classes provides necessary CoABS Grid functionality. For instance, the AgentRegistrationHelper class, as its name implies, assists us in registering our RepeatListener in the Grid. The AgentRep class is the "well-known interface" that all agents within the Grid implement. We use the AgentRep as part of the ServiceTemplate (re: query) to help define the service (in this case an agent) we require. The Message and BasicMessage classes also are intuitive. They both make it possible to send and receive messages between agents. In this case, we use them to send messages to the RepeatBroker. The MessageListener class is an interface used by agents to be notified of incoming messages. The MessageListener interface defines one method: messageAdded(msg:Message). It precludes agents from polling a queue (i.e., MessageQueue). The last class in this group is the CoABSAgentDescription, and as its name implies, it uses public fields to describe various attributes of an agent. The RepeatListener uses one instance of this class to define its own attributes, such as its name, description, ontologies used, agent communication languages used, etc.[55]

The next set of imported classes provides input/output functionality. In sum, they allow the RepeatListener bind to a port, input the text stream, and output it as a file to the disk drive.

[54] Code in this chapter should be considered incomplete. We include primarily code necessary to convey the agent aspects.

[55] Our agents do not implement agent communication languages at the time. Instead, we use the "NaturalLanguage" attribute to indicate we are only passing string values between agents.

After some initial declarations, we find the constructor for this class. We observe that it accepts the agent name as a parameter. In the case of the RepeatListener, we have “hard-coded” it simply enough to “RepeatListener.” Next, we register the agent with the grid, using the agent name we just assigned. Afterward, the agent’s attributes are defined, and then a ServiceTemplate is constructed. Recall, we use the ServiceTemplate class to essentially build a query that returns the required agent proxy. Notably, part of this template includes the AgentRep.class interface. When looking for agents on the grid, this interface is always used.[56] In this case we require the RepeatBroker agent.

With the SearchTemplate built, we now contact the Lookup Service to request the RepeatBroker’s proxy. Next, we bind a server socket (port 2021) and listen for REPEAT-generated OTH Gold text messages. When a message is intercepted, we process it using the process() method. The process method builds the text file and saves it to disk. After the file is saved, the process() method passes the name of the file to the OTH2XML class which transforms the text file to an XML representation. The XML file is eventually stored in the Xindice database in the TempStorage collection (by the XML2Xindice class).

After the file is processed the process() method returns the stored XML file name. If the Lookup Service returned an item from our earlier request (reg.getDirectory().lookup(serviceTemplate) method), we construct and send a (Basic) message to the returned service item (i.e. Repeatbroker). The message includes, among other things, the name of the saved XML file.

This effectively completes a “cycle” for the RepeatListener. There are two other methods in this class, namely the main() method and the messageAdded() method. The main() method simply instantiates the agent, passing it the name; and the

[56] Recalling the Distributed Computing chapter, we can also publish and/or subscribe to services on the grid. If we required a service, we would use that service’s well-known interface, which may likely be unique to the particular service required. This implies foreknowledge of the service is required – shortcoming we hope to solve using SWEB technologies.

messageAdded() is implemented, but not used. As the application grows in sophistication, the RepeatListener will likely receive incoming messages; accordingly, it is implemented as a marker.

The other two agents in this team perform similarly, as it relates to CoABS grid functionality. There specific goals are different, but after understanding the basic Grid and Jini classes used in the RepeatListener, one will be able to analyze the other agents with not too much difficulty.

```
package navy.nps.archangel.repeat.listener;
```

Declare class in listener package

```
import net.jini.core.entry.Entry;
import net.jini.core.lookup.ServiceItem;
import net.jini.core.lookup.ServiceTemplate;
```

Import core Jini classes used

```
import com.globalinfotek.coabsgrid.AgentRegistrationHelper;
import com.globalinfotek.coabsgrid.AgentRep;
import com.globalinfotek.coabsgrid.BasicMessage;
import com.globalinfotek.coabsgrid.Message;
import com.globalinfotek.coabsgrid.MessageListener;
import com.globalinfotek.coabsgrid.entry.CoABSAgentDescription;
```

Import required grid classes

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Calendar;
```

Import various input/output classes

```
/**
 * ArchAngel project
 * RepeatListener listens to a port (currently 2021) to receive GCCS-M
 * (OTH Gold) text messages from the SPAWAR program called REPEAT. The
 * RepeatListener imports the message as a text file and converts it to
 * an XML file.
 * @author Doug Horner, Sam Chance
 */
public class RepeatListener implements MessageListener {

    String brokerName = “RepeatBroker”;
    String XMLFilename;
    private AgentRegistrationHelper reg;
```

Define class variables

```
public static final short PORT = 2021;

/**
 * Constructor. Instantiates an AgentRegistrationHelper to register
 * with the grid. Creates a search template to match items that
 * have a CoABSAgentDescription with name in brokerName. Retrieves
 * proxy for RepeatBroker fm Lookup service. Binds to ServerSocket
 * and listens for messages.  When a message is received, it calls
 * the process() method and then sends a message with a file that
 * contains an XML file name.
 * @param String agentName
 */
public RepeatListener( String agentName ) throws IOException {

    reg = new AgentRegistrationHelper( agentName );
    reg.addMessageListener( this );
```

Facilitates agent grid registration

```
    reg.registerAgent();
```

Register RepeatListener in Grid

```
    System.out.println( “Registered as “ + reg.getName());
```

Define Repeat-Listener Attributes

```
    //Returns agent description to allow setting agent attributes
    CoABSAgentDescription desc = reg.getCoABSAgentDescription();

    String[] ont = {“OTH_Gold”};
    String[] acls = {“NaturalLanguage”};
    String[] contentlang = {“NaturalLanguage”};

    //Assign all desired attribute values to public fields
    desc.ontologies = ont;
    desc.acls = acls;
    desc.contentLanguages = contentlang;
    desc.description = “Listens on port 2021 for Repeat transmissions.”;
    desc.organization = “Naval Postgraduate School”;
    desc.architecture = “ArchAngel”;
    desc.documentationURL = “https://poweredge.nps.navy.mil/route”;
    desc.displayIconURL=http://poweredge.nps.navy.mil:8080/ArchAngel.jpg”;
```

Construct search template to retrieve broker proxy

```
    // Find Broker agent Use well-known (Agent) interface
        and agent
    name (RepeatBroker).
    Class[] classArray = { AgentRep.class };
    CoABSAgentDescription brokerTemplate = new
    CoABSAgentDescription();
    brokerTemplate.name = brokerName;
    Entry[] brokerTemplateArray = {brokerTemplate};
```

Build service template to retrieve RepeatBroker from LUS

```
    ServiceTemplate serviceTemplate =
        new ServiceTemplate(null, classArray, brokerTemplateArray);

    ServiceItem[] items= reg.getDirectory().lookup(serviceTemplate);

    System.out.println(“(Broker) items found: “ + items.length);

    ServerSocket sock;
    Socket clientSock;
    int counter = 1;


    try {
        //New server socket on port 2021
        sock = new ServerSocket(PORT);
```

Listen on port 2021

```
        while ((clientSock = sock.accept()) != null) {

            //Process the OTH Gold text message
            String othMessageName = process(clientSock, counter);
```

Call to process text file

```
            System.out.println(“The message name being sent to the
                    RepeatBroker is: “ + othMessageName);

            //Conditionally, send message to RepeatBroker. Pass file
            name as parameter.
            if (items.length != 0) {

              BasicMessage requestMessage = new
              BasicMessage(brokerName, reg.getAgentRep(),
              “NaturalLanguage”, othMessageName);
              ((AgentRep) items[0].service)).addMessage(requestMessage);
            }
            else System.out.println(“Didn't find a Broker.”);
```

If RepeatBroker proxy returned, send message with XML file name

```
              counter++;
        }
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    catch ( Exception ex ) {
        ex.printStackTrace();
    }
}
/**
 * Accepts a Socket connection and an int counter. Intercepts the
 * OTH Gold text message, calls the OTH2XML class to convert text
 * message to XML format.  Returns a String representing the saved
 * file's name.
 * @param Socket   Socket to connect to.
 * @param int      Counter.
 * @return String  Saved XML file name.
 * @throws IOException, Exception
 */
public String process( Socket s, int counter ) throws IOException,
        Exception {
```

Process intercepted text

```
    System.out.println(“Accept from client” + s.getInetAddress());

    InputStream is = s.getInputStream();
    InputStreamReader isr = new InputStreamReader(is);

    BufferedReader br = new BufferedReader(isr);
    StringBuffer bs  = new StringBuffer();
```

```
        File othDataFile = new File(“c:/ArchAngel/temp/txt/oth” +
            counter + “.txt”);
        counter++;
```

Text file created and saved

```
        PrintWriter out1 = new PrintWriter( new BufferedWriter(
            new FileWriter(othDataFile, false)));

        while (br.ready() {
            bs.append(br.readLine());
            out1.flush();
            bs.delete(0, bs.length());
          }


        OTH2XML oth = new OTH2XML(othDataFile);
```

OTH text file passed to OTH2XML to be transformed to XML

```
        XMLFilename = oth.getMsgName();
        s.close();
        return( XMLFilename );
    }
```

Saved XML file name passed to RepeatBroker

```
    /**
     * MessageListener method to notify RepeatListener a message has
     * been added to a message queue for it.
     * @param msg  Incoming message.
     */
    public synchronized void messageAdded(Message msg) {
        System.out.println(“received response”);
        System.out.println(msg);
        reg.removeMessage(msg);
        System.exit(0);
    }
```

Allows RepeatListener to receive messages

```
    /**
     * Main method instantiates RepeatListener with name
     * 'RepeatListener'.
     * @param args  Not used for this class.
     */
    public static void main(String[] args) throws IOException {

        if (args.length > 0 ) {
            System.err.println(“Syntax: java RepeatListener”);
        }
        else new RepeatListener(“RepeatListener”);
    } //End of main method
} //End RepeatListener class
```

Instantiate RepeatListener

4. (PR) Message Agents[57]

a. Overview

During initial development of the ArchAngel prototype, emphasis was placed on a pre-mission tasking and planning scenario as it might relate to a Personnel Recovery (PR) operation[58]. To demonstrate the agents' ability to collect and process information, we developed the Message agents. As we will see, they proved apt in this effort.

In any mission there is an operational continuum. This is the cycle a military planner goes through in the tasking, planning and execution of a mission. It can be broadly broken into the following phases: pre-mission planning, insertion, infiltration, actions at the objective area, extraction, post-mission analysis and reporting. Software agents can be effectively utilized to help military commanders make better decisions through each phase. The Message agents were developed and employed to assist in the acquisition, extraction, composition, and presentation of valuable, relevant information. More specifically, the Message agent team functionality consists of retrieving information from various USMTF message sources, searching for and parsing incoming information, importing information into a master operational context (MOC) document, and rendering the specified information in a 3-dimensional (3-D) presentation using X3D/GeoVRML[59] inside a (Netscape) browser. The MOC is intended to be a domain instance document, marked up in OWL and stored as part of a background knowledge base (KB).

For the Message agent team we used a set of 9 agents. These broke down into the following types of agents, based on the triad pattern: there was one Message Watch agent (MsgWatch), one Message Broker agent (MsgBroker), and six Message Handler agents (MsgHandler). In addition to these three types of agents we built an

[57] This section, documenting the (PR) message agents, is derived from a report written by Mr. Doug Horner.

[58] The PR mission area was formerly called Combat Search Air Rescue (CSAR).

[59] X3D is an XML-based 3-D rendering technology [http://www.web3d.org/x3d.html]; GeoVRML is a Java-based 3-D mapping specification [http://www.geovrml.org].

interface, or Display agent (DisplayAgent). The Display agent allowed the human operator to render the processed information in a three-dimensional (3-D) presentation using a (Netscape) browser.

The MsgWatch agent was responsible for viewing incoming messages to see if any pertained to the PR mission. To discern the applicable messages, the MsgWatch agent was encoded with the pertinent message types. If a relevant message was observed, the Watch agent sent a notification message to the Broker agent. For the first implementation, the MsgBroker used simple string matches to link the message type with the corresponding Message Handler agent. The MsgBroker agent was responsible to contact the correct MsgHandler agent and notify it that there was a message ready for retrieval from the message-processing center (Xindice). Our initial design included a MsgHandler agent for each type of message. Each MsgHandler agent downloaded the applicable message from the XML database; parsed the incoming message; and stored the parsed message in the knowledge base. Interacting with the DisplayAgent, a human operator could display the composed 3-D scenario in the Internet browser.

In military operations, one primary source of initial information is received through USMTF message traffic. For pre-mission planning of a PR operation there are several messages that give the responding unit a point of departure for planning. Table 6 below describes some of the pertinent messages.

Message Short Title	Message Name	Description
WARNORD	Warning Order	Notification to prepare for mission tasking. (Not used in scenario.)
OPORD	Operations Order	Standard five paragraph order. Transmits instructions/directives to subordinate/supporting military organizations
ATO	Air Tasking Order	Tasks air missions. Cross-force tasking. Intra-service tasking.
SPINS	Special Instructions	Addendum to ATO. Normally provides PR instructions.
INTSUM	Intelligence Summary	Enemy unit information (e.g., strength, location)
SEARCHPLAN	Search Action Plan	Designates actions required from participating search and rescue units and agencies during PR mission
AIRORD (fictitious)	Air Order	Indicates route, Racetrack and control points in the Air Operations Area
SAFER	Situated Area for Evasion and Recovery	Designates locations for potential rescue and recovery actions
SARIR	Search and Rescue Incident Report	Reports any situation that may require a PR operation. (Not used in scenario.)

Table 6. Domain Messages.

All messages, less the WARNORD and SARIR, were developed in XML for our exemplar. The full versions of the messages were stored in Xindice, the open source native XML database we described in the Data Sources chapter.[60] Although we composed more or less complete messages for our (fictitious) scenario, they contained excess data or data not germane to our PR mission.[61] For this reason it was necessary to parse the messages before entry into the background knowledge base. More specifically, we parsed the essential elements of information (EEI) required for our operation. And because the messages were structured in XML, they were easily parsed using XSLT.[62] This illustrates an example of delegating work to agents. That is, instead of manually reading the messages to glean the EEI, we allow the agents to do it for us. With the EEI extracted, the MOC was ready to be populated with domain state information.

After the EEI were stored in the MOC, they were ready to be displayed as 3D graphics in a browser. The presentation was a geographic depiction of the various units captured in the MOC. Simply described, we took the EEI provided by the USMTF messages and implemented this as an overlay to a three-dimensional terrain visualization. Specifically, it includes the following information:

- Target locations
- Enemy positions
- SAFE areas
- Spider routes
- Air control points
- Air control racetracks
- Air routes
- Areas of operation

[60] Recall, when working with XML data and Xindice, there is no mapping between different data models. One simply designs his data model as XML and stores it as XML. As discussed in the Data Sources chapter, this provides tremendous flexibility.

[61] This is typical of most messages and operations.

[62] While messages currently are coded in a text-based format (USMTF), a message encoded in XML is not a large leap. There is an effort underway called the "Joint-NATO XML-MTF Initiative" which has published draft recommendations for encoding MTF messages in XML. [https://www.nctsi.navy.mil/secsite/webmgr/fouo/det/ introinit.asp] (Site is password protected),

- Joint Special Operations Area
- Air Operations Area

Because the information is updated daily via subsequent (USMTF) messages, visualization of the area of operations can be effective for units on standby for a downed pilot response. This can give all participants a better understanding of the PR domain.

Technically this was accomplished as follows: The Display agent was developed to retrieve the MOC and transform its contents to a 3D representation. The 3D representation was accomplished using the Extensible 3D Graphics (X3D) specification[63]. Within the X3D scene, GeoVRML[64] was used to combine terrain images with elevation data (Digital Terrain Elevation Data – Level 1) to produce a quad-tree, 3D terrain representation of the operating area. To produce the overlay in the X3D scene, the agent converted the MOC using two XSLT style sheets. This used the JDOM[65] API and the Java Extension (Javax) package to read in XML and apply XSL Transformations to produce the Virtual Reality Modeling Language (VRML97) scene. We viewed the 3D scene using an Internet browser (Netscape 4.79) with a 3D plug-in called Cosmo.[66] This marked the end state for the Message agent set.

b. Functional Flow

We now discuss a single thread of operation to demonstrate how the Message agents function. As already described, relevant messages were stored in our XML database. From a practical point of view, the database was our message processing center. A more sophisticated construct would allow incoming messages to be stored into the database as they were received from external providers (much like we will see with the REPEAT agents). Indeed, this idea is conveyed below in the functional flow diagram (Figure 32). With the messages stored we initialize the agents. The MsgWatch agent immediately begins comparing message names stored in the database, looking for (string)

[63] See [http://www.web3d.org/x3d.html] for information about X3D.

[64] See [http://www.geovrml.org/] for more information about GeoVRML.

[65] JDOM is a Java-based solution for accessing, manipulating, and outputting XML data from Java code. See [http://www.jdom.org] for more information.

[66] See [http://www.cai.com/cosmo/] for more information.

matches. For example, the MsgWatch looks for an ATO message, and, if found, it sends a notification to the MsgBroker agent that a relevant message was found. Upon receipt of the notification the MsgBroker agent determines what message has been located, and then alerts the MsgHandler. The MsgHandler uses a class to retrieve the appropriate message from the repository. Once retrieved, the EEI are extracted from the message using XSLT. Then the EEI are imported into the MOC also using XSLT. This process occurs for all the pertinent messages in the database.

Once the MOC is completely populated with all declared EEI, the user may elect to render a 3D presentation of the MOC in an Internet browser. It is conceivable that the incoming message could be used to keep the MOC current, and subsequently the display in the browser – all by the agents. Effectively, we may use the agents to show a continually updating "movie" of the Area of Operations.

The functional diagram below illustrates the interactions among the Message agents. The emphasis is on an ATO message that is processed through the Message agent team. A couple of items to note include the following. First, the diagram includes a screen capture of a Xindice browser. The Xindice browser is simply a GUI application able to display collections and documents stored in Xindice. Secondly, similar to the REPEAT function flow figure, we illustrate a computer titled Apache Tomcat. In this instance we used Tomcat to host web page that served as our message portal. A screen capture of the web page we developed to enumerate the messages stored in Xindice is displayed as well. Finally, in the upper right of the figure below, we observe a snippet of XML from the MOC, which feeds into our X3D display.

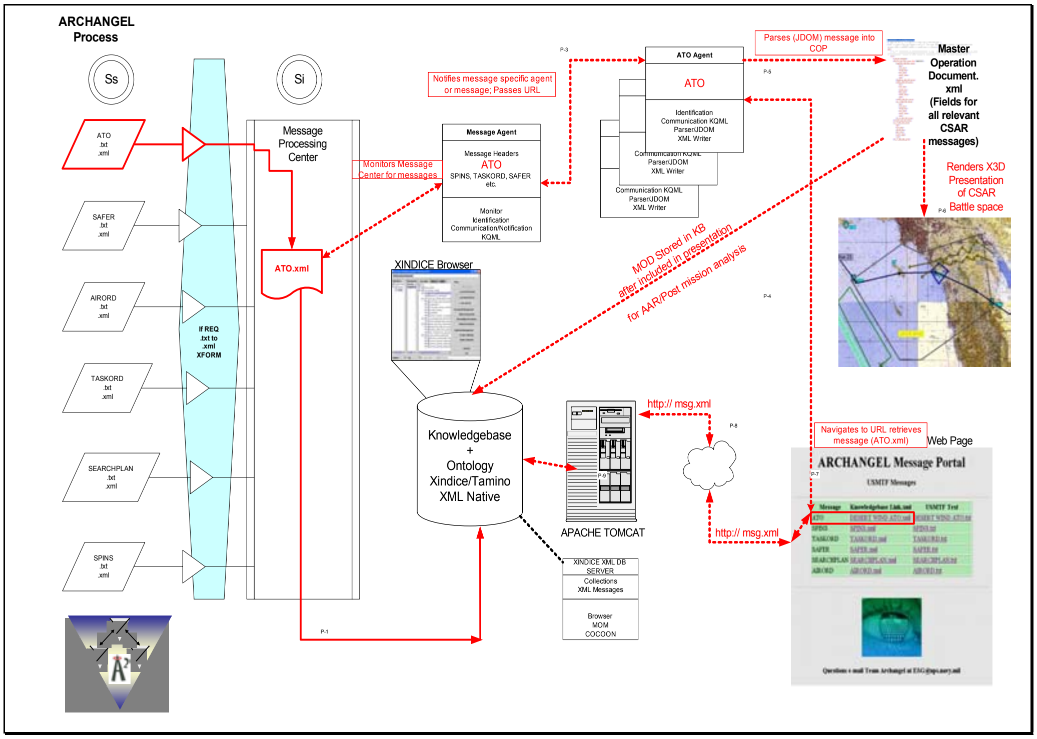


Figure 32. Message Agents Functional Flow.

c. Control Flow

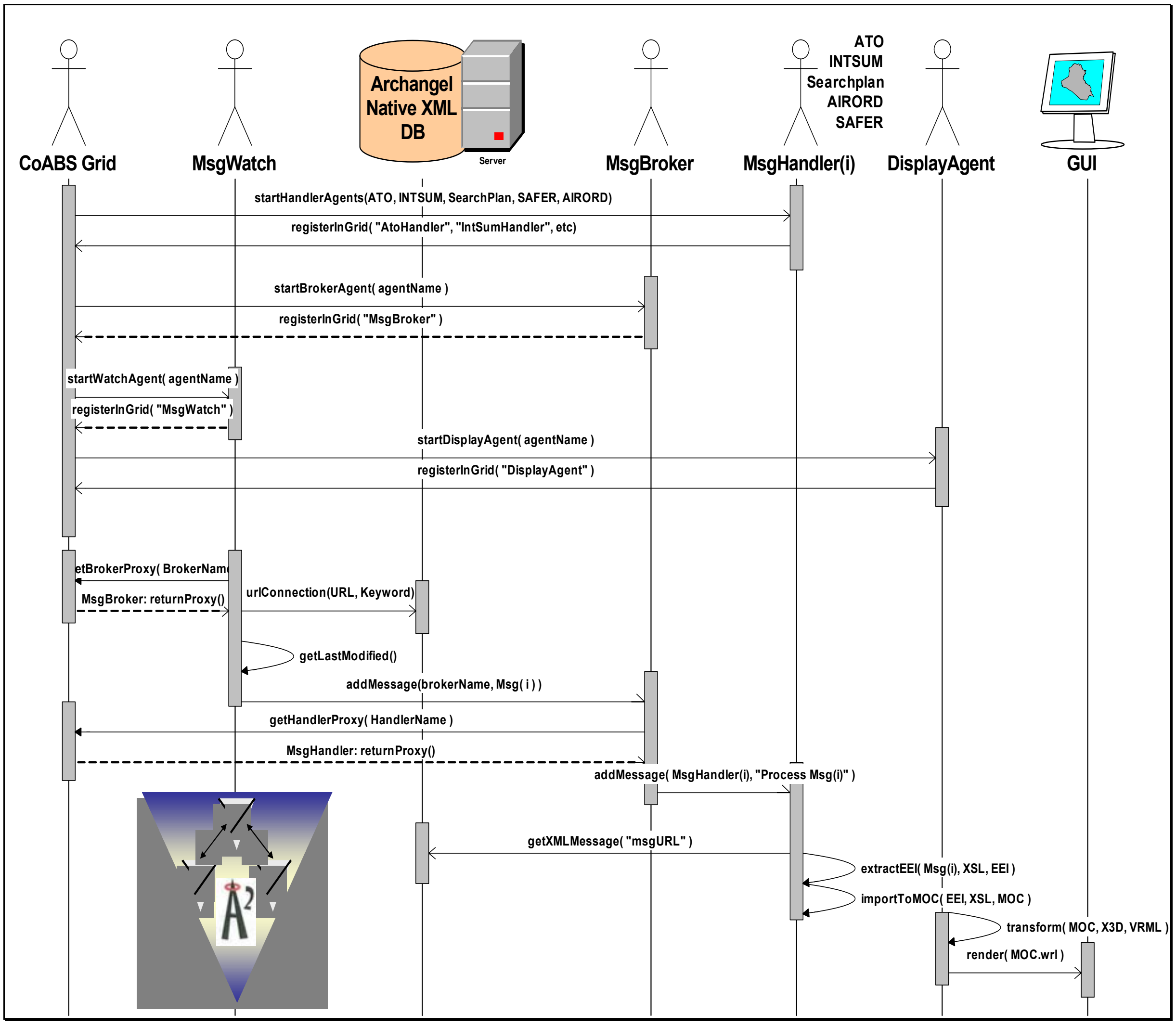
This section provides a more technical survey of the control flow inside and among the Message agents. Similar to the REPEAT agents, we walk through a System Sequence Diagram (SSD) (See Figure 33). The SSD provides a useful mechanism for tracing the control flow of the Message agents. As we observe, the process is begun by initializing the MsgHandler classes and registering them on the grid. Next, the MsgBroker is initialized and registered. Now that these two are active, we initialize and register the MsgWatch agent.[67]

With all agents ready, the MsgWatch agent first builds a SearchTemplate to retrieve the MsgBroker's proxy from the lookup Service. It then connects with the database and checks the last modified date of the messages. The purpose is for the MsgWatch to look for the latest version of the relevant messages. Next, for each applicable message in the database, the MsgWatch agent communicates with the

[67] It is important to note the order we initialized these agents. Due to the way they were initially designed, the MsgWatch agent will begin looking for messages immediately. If the other agents are not registered already, no communication will occur between the agents. This limitation is easily overcome by implementing additional functionality (e.g., "Event handling"). We simply built this design for brevity and rapid prototyping.

MsgBroker agent (proxy) to notify it of waiting messages. Upon receipt of notification from the MsgWatch agent, the MsgBroker agent contacts the designated MsgHandler agent for each message in the database. The MsgHandler then retrieves each pertinent message from the database using message "retriever" classes, and then performs two XSL transforms on the message. The first transformation extracts the EEI; whereas, the second imports the EEI into the MOC. After each MsgHandler has completed the transforms, the MOC is considered refreshed, or current. With an updated MOC the user may call on the DisplayAgent to render the MOC contents in 3D using the internet browser. This completes the Message agents' process.

The Message agent prototype is just a beginning. Over time it could be made much more sophisticated and robust. For example, instead of string matches, we could implement an Agent Communication Language (ACL) to make the inter-agent communications more complex and rigorous. Additionally, the agents could operate in this manner indefinitely, as opposed to cycling through once. Finally, this sort of functionality could be made available as a service to others to use as needed, when needed. In the next section we highlight code from the MsgHandler agent to provide a more detailed look at how the agents are constructed.



CoABS Grid
MsgWatch
Archangel Native XML DB
Server
MsgBroker
ATO
INTSUM
Searchplan
AIRORD
SAFER
MsgHandler(i)
DisplayAgent
GUI
startHandlerAgents(ATO, INTSUM, SearchPlan, SAFER, AIRORD)
registerInGrid("AtoHandler", "IntSumHandler", etc)
startBrokerAgent(agentName)
registerInGrid("MsgBroker")
startWatchAgent(agentName)
registerInGrid("MsgWatch")
startDisplayAgent(agentName)
registerInGrid("DisplayAgent")
etBrokerProxy(BrokerNam
MsgBroker: returnProxy()
urlConnection(URL, Keyword)
getLastModified()
addMessage(brokerName, Msg(i))
getHandlerProxy(HandlerName)
MsgHandler: returnProxy()
addMessage(MsgHandler(i), "Process Msg(i)")
getXMLMessage("msgURL")
extractEEI(Msg(i), XSL, EEI)
importToMOC(EEI, XSL, MOC)
transform(MOC, X3D, VRML)
render(MOC.wrl)
A²

Figure 33. Message Agents System Sequence Diagram.

d. *Example Code*

Below is a code snippet from one of the Message Agent classes (MsgHandler). It first declares this class' package name (i.e., navy.nps.archangel.message.handler). Next it imports the classes that provide the necessary CoABS Grid functionality. For instance, the AgentRegistrationHelper class, similar to the REPEAT agents, assists us in registering our MsgHandler agent in the Grid. The reader may notice we imported no Jini classes in the MsgHandler agent. This is attributed to the fact that this agent does not require a proxy of any sort from the Lookup service. As a matter of fact, it only receives messages from a sending agent (i.e., the MsgBroker). Next we arrive at the class declaration and notice that it implements the

CoABS MessageListener interface. Recalling the REPEAT example, the MessageListener interface defines only the messageAdded() method, and precludes the agent from polling the MessageQueue for incoming messages.

The first functionality we see inside the class definition is a series of MsgRetriever declarations. There is one MsgRetriever class used for each type message we require. Its purpose, as one might imagine, is to fetch, or retrieve, the relevant message from the database. In addition to the MsgRetriever declarations, we declare two (XSL) Transform objects. These will be used to apply XSL style sheets to the different messages. Finally, an AgentRegistrationHelper is declared to facilitate agent registration in the Grid.

After some initial declarations we find the constructor for this class, and observe that it accepts the agent name (delegateName) as a parameter. In the case of this MsgHandler, we have "hard-coded" it to "AtoHandler." Next, we register the agent with the grid using the agent name we just assigned, and add the message listener so it will "hear" incoming messages. With the MsgHandler initialized and registered, we are ready to process incoming messages.

Inside the messageAdded() method we first get the raw text of the message. Then we check to see if the raw text matches on of our message types, by comparing it to a string value. If we find a match, we get the message from the database using the designated MsgRetriever class, and then save it to file. After the file is saved to disk, we perform our two transformations.

The first transformation uses, in the case of an ATO message, the ATO.xsl to extract the EEI from the ATO.xml message file. The resulting, or output, file is called AtoMOC.xml, and it is used to import ATO EEI into the MOC document. This is accomplished by the second transform which uses the current MOC.xml file and the Ato2MOC.xsl style sheet to import the EEI. The resulting file is the (updated) MOC.xml.

The "else if" statement is repeated for each of the message type (e.g., ATO, INTSUM, etc.); however, for brevity, the remaining sections were not included in the code snippet. Each section calls separate message retriever instances and uses separate stylesheets to refresh the MOC. The main() method serves the same purpose as it did in the REPEAT code snippet; that is, it instantiates the agent using the coded name. (i.e., AtoHandler).

```
*
*  ArchAngel project
*  MsgHandler does the following:
*
*  1. Receives communication from the MsgBroker Agent saying that a
message
*     is ready for download
*  2. Downloads the message by instantiating the MsgRetriever class
*  3. Uses JDOM and XSLT to parse the message into the key elements
*     (XSLTransform)
*  4. Uses JDOM and XSLT to transmit the information into the Master
*     Operation Context document
*
*/
```

Package declaration

```
package navy.nps.archangel.message.handler;
```

Import grid functionality

```
import com.globalinfotek.coabsgrid.AgentRegistrationHelper;
import com.globalinfotek.coabsgrid.Message;
import com.globalinfotek.coabsgrid.MessageListener;


public class MsgHandler implements MessageListener {
```

Declare MsgRetriever for each message type

```
  MsgRetriever AtoMsg, IntsumMsg, AirordMsg, SearchplanMsg, SaferMsg;
  XSLTransform transform, transform2;

  private AgentRegistrationHelper reg;

  public MsgHandler(String delegateName) throws IOException {
```

Register handler agent

```
    reg = new AgentRegistrationHelper(delegateName);
    reg.addMessageListener(this);
    reg.registerAgent();
  }

  public synchronized void messageAdded(Message msg)  {
    try {
      System.out.println("******* RECEIVED MESSAGE *******\n" + msg);

      String MsgHeader = new String(msg.getRawText());
      if (MsgHeader.compareTo("Process ATO message") == 0 )
      {
         //Get message from the URL
```

Get ATO and save to disk

```
         AtoMsg = new MsgRetriever();

      AtoMsg.sGetXMLMessage("http://131.120.179.192:8080/xincon/db/usmtf_mes
             sages/ato/ato_302100Oct2002_.xml","C:/ArchAngel/agents/ATO.xml");
```

Extract EEI

```
        //Transform ATO.xml into the parsed document for entry into the MOC
        transform = new XSLTransform(“ATO.xml”, “ATO.xsl”, “AtoMOC.xml”);
```

Import EEI to MOC

```
        //Open the MOC file and append it using the ATOMOC.xml document
         transform2 = new XSLTransform(“MOC.xml”, “Ato2MOC.xsl”, “MOC.xml”);

        System.out.println(“MOC.xml updated with the latest ATO info”)  }

     else if (MsgHeader.compareTo(“Process INTSUM message”) == 0 )
     {
```

Same as above.

```
        //Get message from the URL
        IntsumMsg = new MsgRetriever();
        IntsumMsg.sGetXMLMessage(“http://131.120.179.192:8080/xincon/db/
            usmtf_messages/intsum/intsum_300902Oct2002_.xml”,
            “C:/ArchAngel/agents/INTSUM.xml”);

        //Transform ATO.xml into the parsed document for entry into the MOC
        transform = new
        XSLTransform(“INTSUM.xml”,”INTSUM.xsl”,IntsumMOC.xml”);

        //Open the MOC file and append it using the ATOMOC.xml document
        transform2 = new XSLTransform(“MOC.xml”,”Intsum2MOC.xsl”,”MOC.xml”);

        System.out.println(“MOC.xml updated with the latest INTSUM info”);
     }

        //Code elided.

     .
            .
            .
   //The “else if” construct is repeated for each message type we are
interested in retrieving and transforming.

     }
    catch (Exception exc) {
      System.out.println(“Exception in messageAdded(): “ + exc);
    }
  }

  /** Main program.  Instantiates an agent.
   **/
```

Instantiate handler agent

```
  public static void main(String[] args) throws IOException {
    if (args.length < 1)
       System.err.println(“Syntax: java MsgHandler AtoMsgHandler”);
    else new MsgHandler(args[0]);
  }
} //end class
```

E. SUMMARY

Our work developing agent prototypes demonstrated many of the concepts discussed in the first part of the chapter. For example, Jini, built on Java technology, provided us code mobility, and inter-agent communication. The Grid made it easier to build and deploy agents who could be stored in a Lookup service and could find other agents. Our research is just a beginning, and our agents only achieve simple tasks; still we have demonstrated the relative ease at which agents can be developed and deployed to support military operations. To this end, we have demonstrated support to military operations in the following respects.

- Enhanced situational awareness
- Information from macro-level environments (GCCS-M) reduced and aggregated to a mission specific scenario (i.e. Personnel Recovery)
- Operational Modeling and Simulation
- Delegation of tasks to software processes (agents) freeing humans to concentrate on more complex responsibilities.

Future work for our agents includes converting the REPEAT agents into an XML Web service and/or OWL-S.

Given the fundamental requirement to achieve and maintain knowledge superiority, agent implementations will pervade. We look forward helping to realize the enhanced capability agents will deliver the war fighters.

The key points of this chapter relate to the promise of agents and their significance to the SWEB. The first part of the chapter described many of the attributes and classifications of agents. The second part illustrated some concrete agents as implemented in our research. Another more subtle point is the relative ease at which agents can be developed and deployed.

Now that we have gained a deeper understanding of agent concepts and have seen concrete examples, we are ready to delve into the knowledge-centric aspects of SWEB components. Specifically, the next chapter discusses ontologies and how they specify or frame knowledge. After we examine ontological concepts and examples, we move on to elaborate on knowledge bases. Agents will prove critical to both of these components in a SWEB application. This is where they will do more than listen, broker, and handle; they will employ more sophisticated intelligence and decision-making capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. ONTOLOGY: FRAMEWORK OF KNOWLEDGE

A. BACKGROUND

> Nobody will ever categorize everything, but many people will categorize
> some of it many times over, often in different and conflicting ways.
>
> -Aristotle

The purpose of this chapter is to highlight the importance of the ontology to the SWEB. We will review the Web Ontology Language (OWL) and analyze the basic components of an ontology. We will highlight ontology design criteria, methodologies and various ontology design patterns. Our discussion will culminate by exposing several of the challenges a developer will face while designing and deploying an ontology for practical use.

There are as many definitions for ontology as there are methodologies and markup languages to construct them. When searching for a widely accepted definition all agree the term is borrowed from philosophy and metaphysics, and most agree with its association to the nature of being and existence (Maedche, 2002, 13). One of the earliest concerned with the "nature of being" and classification of objects was Plato and his student Aristotle (Sowa, 2000, 356) (Maedche, 2002, 13). Aristotle's work focused on classifying subjects into groups using logic[68]. He discovered pure logic had limitations and not all subjects were easily classified (Sowa, 2000, 356).

1. Ontology Defined

Merriam-Webster's Dictionary states language at its basic level is a formal system of signs and symbols organized to form terms committed to by a domain. An ontology at its basic level is a set of terms and associated meaning built on a language or representation. The ontology is both language and meaning (Obrst, 2003, 122). Obrst establishes that "An ontology encompasses both meta data and domain theories," where the metadata describes the semantics [language] and the domain theory establishes the relationships, attributes and constraints [meaning] of the model (Obrst, 122, 2003). By constructing an ontology we are establishing the domain theory which all members of the domain must commit. Commitment is the agreement by members of the domain to

[68] Classification is one of the fundamental purposes of ontologies (Jasper, 1999, 5).

subscribe, reuse and extend the concepts and terms of the ontology. If domain commitment is weak or absent, the ontology will be ineffective. The full potential of the SWEB cannot be realized without commitment. In a sense, we are classifying the objects of a universe of discourse into like groups, with agreement from the members of the domain, similar to what Aristotle attempted to do.

2. Commitment

The importance of commitment extends much deeper than the set of lexical terms used by an ontology. Commitment also extends to the ontology's conceptual grounding. To commit to an ontology means all observable actions[69] are consistent with the definitions prescribed in the ontology, and all members are in agreement to use the shared vocabulary in a coherent and consistent manner (Gruber, 1993, 2). An ontology is an explicit specification of a conceptualization, where a conceptualization is an abstract, simplified perception of the world we wish to represent for some purpose (Gruber, 1993, 1-2) such as use by information systems and their proxies. This use must be enabled from the basic syntax through the taxonomy or organization, to the most complex concepts and processes. Information, knowledge, and web based systems are already committed to some conceptualization upon design, whether explicitly or implicitly (Gruber, 1993, 1). Therefore, to ensure a successful implementation of an ontology, it must be part of the explicit commitment.

Within the military, albeit more standardized than the civilian world, disciplined commitment to a domain theory or theories will be a daunting challenge. However, since the military is built on doctrine, and doctrine is a baseline for many activities, a doctrine based domain theory provides a promising start point for the origination of this explicit commitment. The problem with doctrine is tactics, techniques, and procedures derived from operational experience will always trump the doctrinal baseline. Doctrine is what we read and teach, and not precisely what occurs in "real world" operations. This deviation from doctrine can be termed practical drift.

[69] Actions regard the unambiguous, consistent interpretation and usage of a term or concept described in the ontology. There must be no circumstance in which an action occurs inconsistent or contradictory to its description in the ontology.

For the purposes of this work we will adopt Obrst's definition of an ontology which is a vocabulary expressing entities and relationships of a conceptual model for a general or particular domain along with the semantic constraints on that model which limit what the model means (Obrst, 2003, 122). More succinctly put, ontologies provide the meaning and the context for the domain so computers can understand, or at least interpret meaning from a given set of ontologically defined terms.

B. THE WEB ONTOLOGY LANGUAGE (OWL)

Since ontologies will provide the meaning to the applications of the SWEB, deciding on an ontology language recommendation was a necessary step for the World Wide Web Consortium (W3C). From the requirements, the Web Ontology Language (OWL) emerged as W3C's choice for the ontology language of the SWEB. An ontology language like any other language must be firmly based on practical usage and rooted in proven baseline concepts. Figure 34, The Semantic Wave, demonstrates the solid grounding OWL inherited from proven concepts and technologies such as XML, Resource Description Framework (RDF), Resource Description Framework Schema (RDFS) and especially the DARPA Agent Markup Language + Ontology Interface Layer (DAML + OIL).

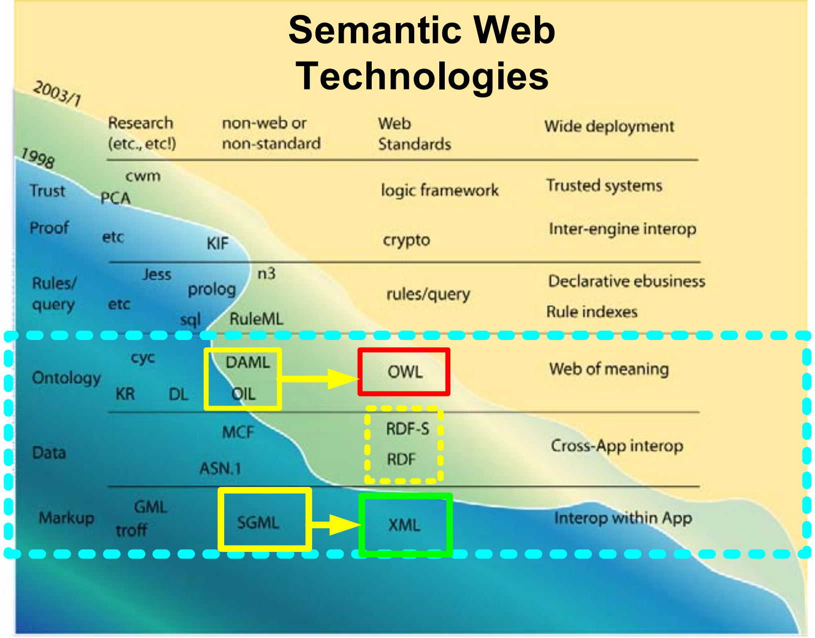


Figure 34. Semantic Wave (After: Berners-Lee, 2003).

The DAML+OIL program was the well established ontology language and predecessor to OWL. DAML+OIL imparted many years of valuable practical usage and implementation experience to the OWL initiative. The maturity and experience of the DAML+ OIL program catalyzed OWL's rapid movement from a working draft to a pending recommendation. These technologies, to the credit of the developers, were not discarded in favor of something new, but their strengths were leveraged and retained as part of the OWL Recommendation.

It is worth mentioning the importance of XML to OWL and the larger SWEB movement. When XML emerged it was considered a "Concept elegant in its simplicity driving dramatic changes in the way Internet Applications were written" (Birbeck, 2001, 1). It has in fact performed as expected and is only now starting to gain wide spread adoption and momentum. Within the Military and DoD XML initiatives are gaining momentum as well. The USMTF Message program is continuing work on establishing XML schemas and formats for all the USMTF messages. The Battlefield Management Language (BML), an unambiguous language derived from XML used to control forces and equipment, is also being developed for the Army (Carey, 2002, 1). Behind the scenes XML is starting to reach critical mass. Just as XML has started to prove its value to industry, the developers of OWL leveraged XML's stability and functionality incorporating it into OWL. XML is one of the foundations from which the SWEB will be built.

Figure 35, the Semantic Layer Cake, provides a more intuitive view of the foundational technologies and their contributions to the OWL language. This figure demonstrates how the supporting technologies have contributed to the finalization of OWL, as OWL is built on top of their existing and functional foundations.

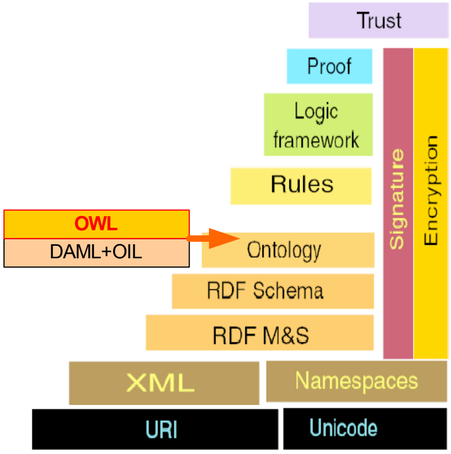
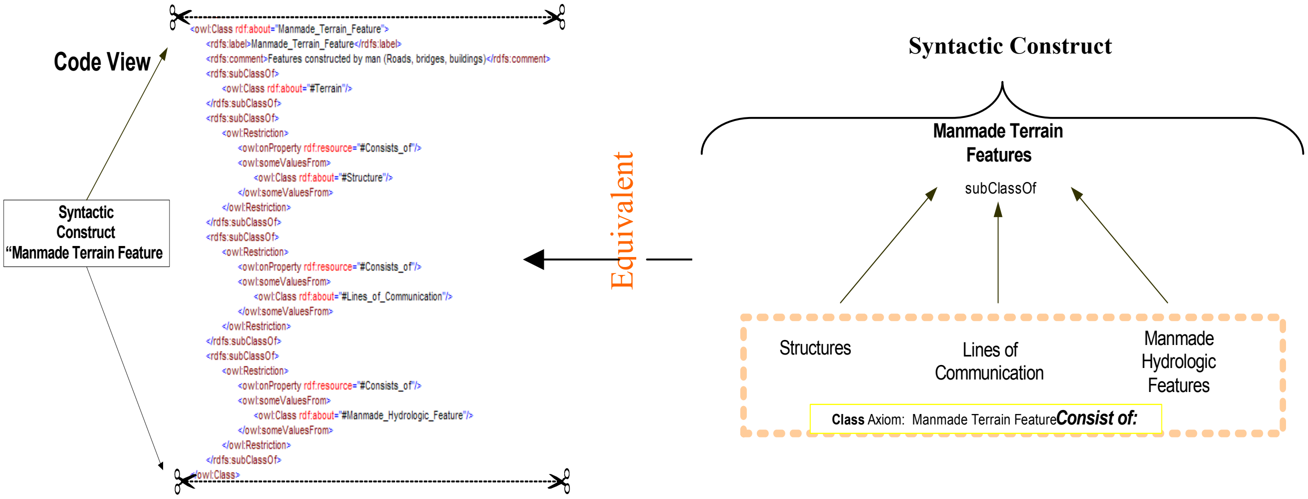


Figure 35. Semantic Layer Cake (After: Berners-Lee, 2001).

OWL was designed for use by applications required to process content as opposed to the traditional role of presenting it to human users (McGuinness, 2003, 1). As we have discussed above OWL takes its foundation from many different proven technologies. OWL at the syntax level adopts a frame-like style, where the information about a class or property is given one large syntactic construct, instead of being divided into several atomic chunks, or triples (Patel-Schneider, 2003, 7). The concepts of class and property will be described at length later. For now, consider both properties and classes as two foundational components of an OWL ontology. The 'large syntactic construct' OWL employs lends an embedded organization or taxonomy within a given OWL ontology. This organization creates a more human readable product. The 'syntactic construct' embedded in OWL is illustrated in Figure 36.

The example ontology used in Figure 36 is taken from a larger ontology in the Tactical Routes domain. A Tactical Route is defined as a planned direction of travel by a small unit through a semi-permissive or non-permissive environment. One of the concepts important to Tactical Routes are descriptions of Manmade Terrain Features. The complete description of the owl:Class Manmade Terrain Feature is located conveniently in one large construct shown in Figure 36. The directed graph on the right illustrates the same syntactic construct exposing the embedded hierarchical structure in visual form. Both constructs are equivalent.

To reinforce this point, we will step through the OWL class description of Manmade Terrain Feature in order to explain its content. The first owl:Class establishes Manmade Terrain Feature as a class in our ontology. Next owl:Class Manmade Terrain Feature is declared a subClassOf: Terrain[70]. The rest of the syntactic construct exhaustively enumerates the members belonging to the Manmade Terrain Feature Class by the restriction on the property Consists of. The restriction states that the members of the Manmade Terrain Feature must be members of the classes Structure, Lines of Communication and Manmade Hydrologic Feature.



```
<owl:Class rdf:about="Manmade_Terrain_Feature">
    <rdfs:label>Manmade_Terrain_Feature</rdfs:label>
    <rdfs:comment>Features constructed by man (Roads, bridges, buildings)</rdfs:comment>
    <rdfs:subClassOf>
        <owl:Class rdf:about="#Terrain"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#Consists_of"/>
            <owl:someValuesFrom>
                <owl:Class rdf:about="#Structure"/>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#Consists_of"/>
            <owl:someValuesFrom>
                <owl:Class rdf:about="#Lines_of_Communication"/>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#Consists_of"/>
            <owl:someValuesFrom>
                <owl:Class rdf:about="#Manmade_Hydrologic_Feature"/>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Code View
Syntactic Construct "Manmade Terrain Feature
Equivalent
Syntactic Construct
Manmade Terrain Features
subClassOf
Structures
Lines of Communication
Manmade Hydrologic Features
Class Axiom: Manmade Terrain FeatureConsist of:

Figure 36. OWL Syntactic Construct.

1. OWL Features

We have now demonstrated how OWL embeds a taxonomic organization into the ontology through the illustration in Figure 36. What other beneficial features does OWL offer its potential users? To demonstrate some of its features let us take a closer look at the OWL language and see why it will be one of the important Knowledge Representation frameworks of the SWEB.

a. OWL Sub-languages

OWL is a flexible language providing different implementation options to users. The family of OWL is decomposed into three sub-languages: OWL Lite, OWL

[70] We will discuss the design of the class hierarchy in detail in a later section. For now we are concerned with only the assertion of concepts and classes within the ontology.

Description Logic (DL), and OWL Full. Each of the OWL sub-languages embodies a trade off between simplicity, decidability[71] and expressiveness. In general, the more simple the construct of the language the least expressive it is. While a gain in expressiveness is considered desirable in some applications, expressiveness sometimes disqualifies any computational guarantee. Figure 37 below shows the expressiveness versus complexity continuum present in the OWL family of languages. In the next section we will discuss each of three OWL sub-languages in more detail.

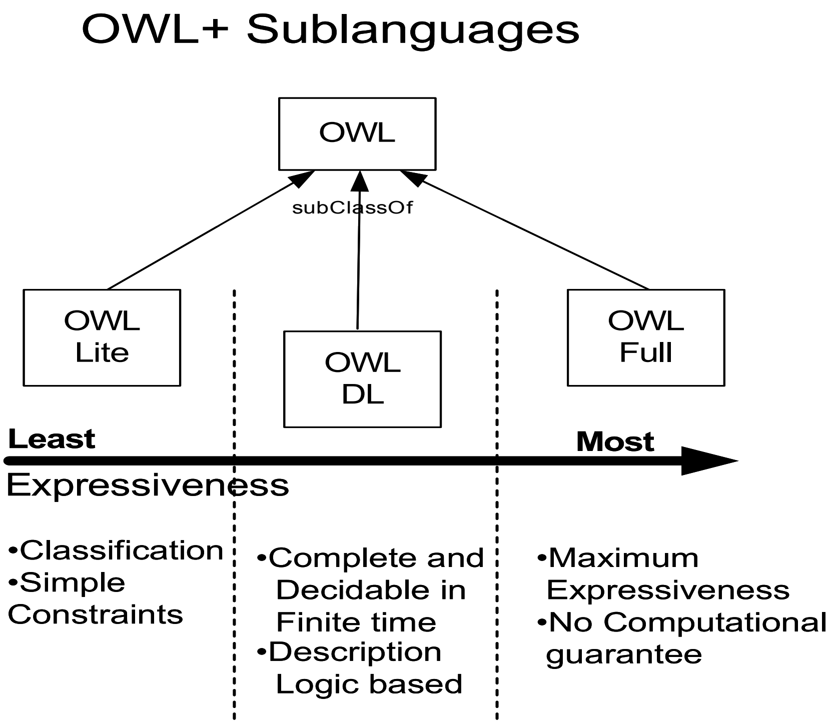


Figure 37. Web Ontology Language Hierarchy.

(1) OWL Lite. OWL Lite should be the choice of users who want a mechanism specializing in establishing a classification hierarchy (McGuinness, 2003, 4). OWL Lite was designed for easy implementation by providing users a subset of constructs with which to become familiar and to catalyze tool development (van Harmelen, 2003, 5). OWL Lite only provides the constructs for a subclass[72] hierarchy

[71] Decidability is defined as the ability to resolve a logic or computational operation in a finite time.

[72] Subclass is a more specific, subordinate (at a lower level) in a class hierarchy.

and limited value cardinality (van Harmelen, 2003, 44). The only cardinality[73] explicitly stated within OWL Lite is 0 or 1. OWL Lite is the simplest to use and least expressive of the OWL sub-languages providing only classification and simple constraints.

(2) OWL DL. OWL DL is an extension of OWL Lite. OWL DL is the choice for users requiring more expressiveness than OWL Lite while retaining computational completeness and decidability for use with reasoning systems (Smith, 2003, 5). OWL DL was designed to support the existing Description Logic business segment (Smith, 2003, 5). OWL DL and OWL Full subscribe to the same vocabulary, but OWL DL has additional restrictions which give it desirable computational properties (van Harmelen, 2003, 43). OWL DL provides the most expressive options while preserving computational guarantees.

(3) OWL Full. OWL Full contains all the constructs of the OWL Language and provides free and unrestrained expressibility (van Harmelen, 2003, 41). OWL Full, unlike OWL DL allows classes to be treated like individuals and allows the ability to differentiate between instances[74] (van Harmelen, 2003, 42). OWL Full has direct mappings to the RDF constructs. For instance, the embedded meta class owl: Thing is equivalent to the top level RDF construct rdf: Resource and owl: Class is equivalent to rdfs:Class. This direct mapping while preserving the meta modeling properties of RDF, causes the loss of any computational guarantee (van Harmelen, 2003, 42).

2. OWL Relationships

Since we have been exposed to the OWL sub-languages let us look more closely at the relations existing between them. To begin, Figure 37 does not accurately capture OWL's true relations and is somewhat misleading. Figure 38 better illustrates the relationships of the sub-languages from an ontology classification and the set of possible conclusions drawn from reasoning against it. Since both threads are similarly interpreted

[73] Cardinality is a numerical restriction on an allowable number or allowable range of numbers. Cardinality can be established for Minimum, Maximum, or Exact values.

[74] An instance is defined as the atomic level of an ontology.

we will not discuss both. The reader may simply substitute conclusion for ontology and derive the same results.

From the ontology perspective every OWL Lite Ontology is a legal OWL DL Ontology (Smith, 2003, 5). In this case, OWL Lite is more specialized by exclusion. OWL Lite assumes its specialization by further constraining the language constructs of OWL DL. As we have stated above, OWL Lite is the simplest, or most constrained, and least expressive of the OWL sub-languages. Another way to view OWL Lite is as the most specialized of the OWL sub-languages. OWL DL follows similarly, as it can be viewed as a more highly specialized version of OWL Full, also by exclusion (Smith, 2003, 5).

When examining OWL Full it is important to note it is not a formal sub-language (van Harmelen, 2003, 41), but is actually an alias for the family of OWL languages. OWL Full contains all the constructs for the OWL language (van Harmelen, 2003, 41) and, therefore, OWL Full is equivalent to OWL if we are referring to OWL as a super class.

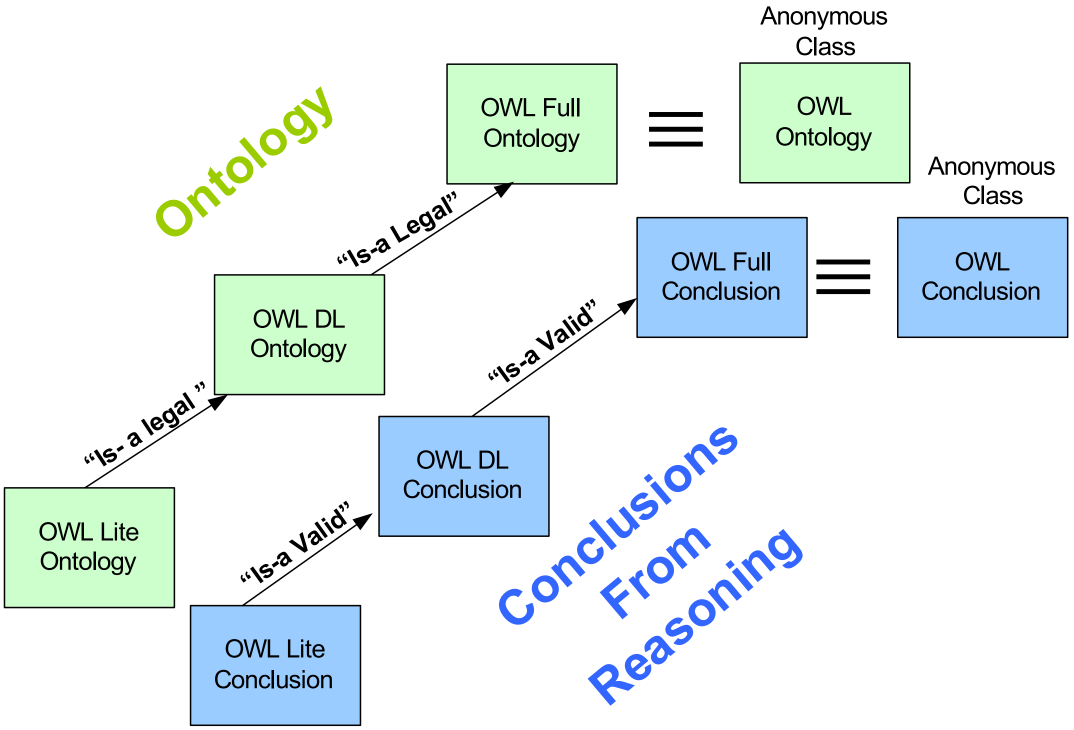


Figure 38. OWL Sub-language Relationships.

3. Description Logic Foundation

The goal of an ontology is not to end with the description of the domain of discourse, but to end with a machine interpretable/computational model of the domain of discourse. To achieve this goal a form of logic programming must provide the underpinnings for the OWL language constructs. This embedded logic is the anchor point to which the concepts and properties are attached enabling the computer to interpret and reason. The OWL language[75] is no exception, and bases itself on Description Logics (DL).

DL is a family of Knowledge Representation (KR) formalisms representing the knowledge of a domain by defining relevant concepts and then using the concepts to specify properties of objects and individuals occurring in the domain (Baader [DL Handbook], 2003,43). DLs have a formal, logic based semantics and support reasoning as a central service (Baader [DL Handbook], 2003, 43). DLs have a model-theoretic semantics and both the concepts and instances usually can be expressed in pure, first order logic[76] (Baader [DL Handbook], 2003, 46). Therefore, a DL language can be considered "first order logic plus", due to the fact it is more expressive than First Order Logic. It is important to understand that OWL's DL underpinnings are abstracted from the user by the OWL's embedded syntax. The purpose is to simplify the construction of ontologies and make it user friendly so domain experts can readily perform this task. For the SWEB to be realized domain experts, not knowledge engineers, must be the ones to construct and maintain the ontologies.

C. COMPONENTS OF AN ONTOLOGY

There are many different terms used to describe the atomic components of an ontology. The terms we will use through out this work are the terms contained in the Web Ontology Language (OWL) Reference (Van Harmelen *et al.,* 2003*,* 9), as this will be the World Wide Web Consortium's (W3C) recommendation for an ontology language. At the time of writing this work the OWL Recommendation was in last call with all indications endorsement was forthcoming. Since the recommendation is pending, we

[75] Includes all sublanguages: OWL Lite, OWL DL and OWL Full.

[76] There are some exceptions. The DL Handbook states that when this is the case some extensions may be applied to the First Order Logic to accomplish the required expression.

consider OWL to be the ontology language most likely used by the military user and developer community. Because of this, we will be discussing ontology components and demonstrating their usage and functions exclusively in terms of OWL. It is important to note our research was not limited to OWL, it is only because OWL is intended to be the standard we give it our focus. However, where applicable the terms and concepts used by the Artificial Intelligence (AI) community will also be addressed as potential synonyms, as they are widely encountered in literature addressing this subject.

Before beginning our discussion on designing an ontology it is necessary to address the components of an ontology from a high level view. Liken this to flying over an ontology and looking down from an aircraft at 10,000 feet, and what you can see will be the subject of the discussion. This high level view is intended to assist us in readily identifying the fundamental components of an ontology and associating basic functions with them. Many will recognize this technique as one used by the Object Oriented (OO) community when decomposing highly complex entities to a set of classes and objects (Booch, 2001, 36). Many of the methodologies to be discussed later will begin by identifying objects in a given domain and classifying them as one of the components of an ontology; therefore, to understand many of the methodologies we must be able to identify an ontology's basic elements.

1. Basic Components of an Ontology

a. Classes

Classes are the fundamental building block of ontologies and the most readily identifiable within a given universe of discourse. Much of the literature attempts to describe classes in a very formal way; however, a class can be roughly determined by asking some pre-competency questions. The first question to ask is, "Do I care about the given concept?" If the answer is "yes" then a follow on question must be asked. The follow on question is, "Do I need to know about this concept for my specific purpose?" If the answer is "yes", it is likely you have discovered a viable class for a given domain. Now apply the formal definitions as a cross check to ensure the potential classes are in fact viable classes and compliant with the formalizations. Now that we have discussed the informal identification of classes, let us continue with a formal definition.

Classes in an ontology, as with the OO methodology, provide a means to classify and abstract resources with similar characteristics (Booch, 2001, 103) (Van Harmelen, 2003, 55). Classes are also referred to as concepts (Noy, 2001, 3) or entities (Obrst, 2003, 125). Classes are the concepts within the domain of discourse that require describing and defining so all other classes can be grounded. Classes can range from general to very specific and highly specialized (Noy, 2001, 6).

To illustrate what we have discussed so far imagine we are interested in creating an ontology for the domain of Military Routes for later implementation into a system required to classify Military Routes into various categories. The subject of Military Tactical Routes is particularly suited for description in the ontology because of the high degree of expertise required to plan and navigate a tactical route successfully. The Tactical Route ontology will capture the view of the expert navigator so the knowledge embedded within can be transferred and shared among non-experts. For now, we are interested in describing the different types of Tactical Routes. Tactical Routes can be a candidate for a class in our ontology. The potential class of Tactical Routes satisfies our pre-competency questions described earlier and meets the formal definition of an ontological class. The class of Tactical Route might be the most general type of class we will identify for our specific interest.

What other types of Tactical Routes are there? What about Dismounted Tactical Routes and Mounted Tactical Routes. Both can be considered Tactical Routes, although they are more specialized. A Mounted Tactical Route implies a route used by vehicles, while a Dismounted Tactical Route implies use by foot traffic. This specialization implies a position lower on the hierarchy than our general class of Tactical Route. These potential classes clearly satisfy our pre-competency questions, so they are good candidates for inclusion in our Tactical Route ontology.

To visualize our newly established classes we can use a directed graph. A general directed graph consists of nodes and arcs. Nodes are the circles and arcs are the lines connecting the circles. Classes will take the role of a node on a directed graph.

Since we have not established the ontological component that fills the role of the arc yet, a directed graph is not of much utility at this point. Nonetheless, Figure 39 illustrates our example thus far.

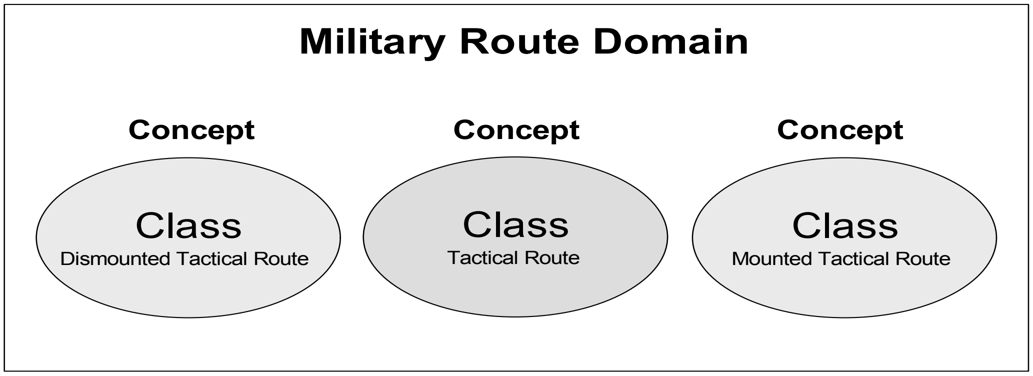


Figure 39. Military Route Class Identification.

In the OWL Recommendation the highest level class or meta-class is predefined for us. In OWL the class “Thing” (owl:Thing) is the top level class of which everything described in OWL is a member (Smith *et al* , 2003, 10). Each time a user declares a class within the OWL constructs it becomes implicitly a subclass of owl:Thing (Smith, 2003, 10). Because OWL has defined the highest level meta class as “all individuals”, the complement or the OWL class “Nothing” (owl:Nothing), or the empty set is also defined for us. This follows the logic that an ontology describes concepts in terms of other defined concepts in the domain. If owl:Thing describes everything, it is quite simple to draw the conclusion that owl:Nothing describes nothing. More formally, “nothing” described in OWL can be a member of the class owl:Nothing.

Classes are further described by developing class descriptions which are used to determine membership to the class (van Harmelen, 2003, 10). The members of a given class are called the class extension. Class extensions can be made up of either instances or other classes depending on the level of abstraction the class represents. These descriptions remind us that classes represent concepts and not the term used as the name for the class (Noy, 2001, 13). The name for the class can change, but the class concept defined by the description should remain the same if the concept is complete. In other words the term should be independent of the concept's meaning. The OWL Reference allows six types of class descriptions (Van Harmelen, 2003, 10):

- class identifier in the form of a URI or URL reference
- exhaustive enumeration of individuals whose collective membership form the class
- property restriction
- intersection of two or more classes
- union of two or more classes
- complement of a class description

b. Subclasses

Now that we have exposure to what it means to be a class of an ontology, we now have to look at the level of abstraction we require the class to possess for our purposes. To accomplish this we can ask another pre-competency question from the top down. "Can the most general classes we have identified be further specified?" If the answer is "yes", and our application will benefit from the increased specificity, then we should establish a subclass of that class, as long as the most specific class "is a" kind of" the most general class (Noy, 2001, 12).

The same is true for the most specific of the classes we have identified taking the bottom up approach. The pre-competency question for this would be, "Is there a generalization that this specific class could belong that we care about?" If the answer is "yes", we should look to further abstract the class, and as long as the most specific class is a "kind of" the most general class, the subclass is valid. Before establishing subclasses we must be sure our application will benefit from either the increased generalization or specialization. If a clear benefit is not evident, then careful consideration should be given before establishing subclasses. The organization of classes will be explained in greater detail in a later section.

In keeping with the directed graph visualization let us update the illustration of our Tactical Route Ontology. In a directed graph, sub classes are also nodes although they are subordinate to the more general classes because they are themselves classes. For our Tactical Route example it is clear both Dismounted Tactical Route and Mounted Tactical Route are specializations of the more general Tactical Route (See Figure 40). Since our application will be charged with classifying various Military Routes we can realize some benefits from this specialization. However, how do we show on a directed graph that these sub classes are related to their parent class? Properties are the answer and they take the form of arcs in our directed graph.

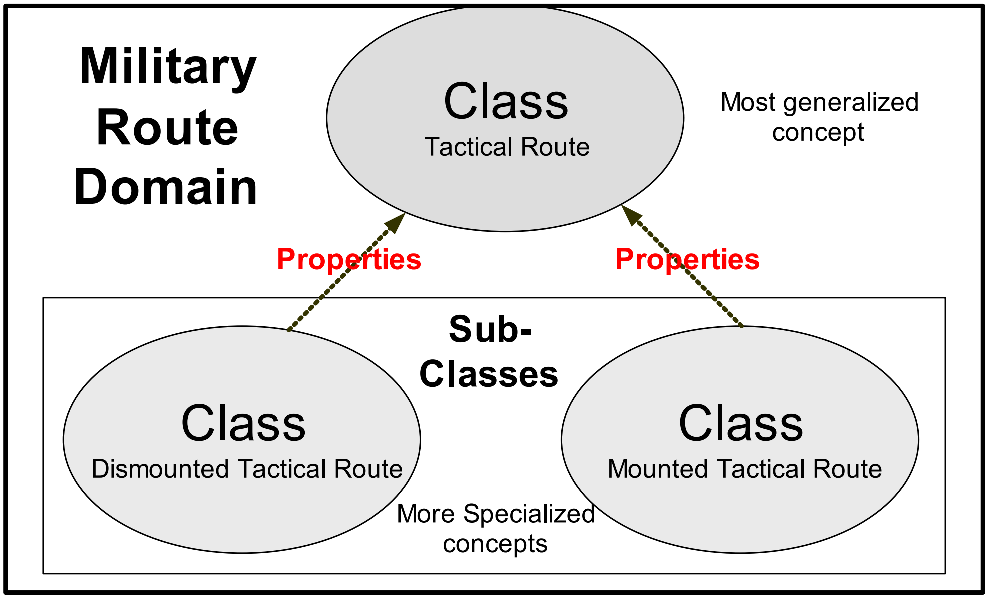


Figure 40. Military Route Sub-Classing.

c. *Properties*

Properties are the glue that connect classes and establish relationships within an ontology. Properties are also referred to as slots or roles within the AI Community (Noy, 2001, 3). Properties establish the internal structure of the concepts (Noy, 2001, 3) and assert facts about the members of the classes (Smith, 2003, 15). Within the OWL constructs there are two types of properties, Object Properties (owl: ObjectProperty) and Datatype Properties (owl:Datatype Property). Object Properties have a value range of class individuals and link instances between two classes (Smith, 2003, 15). Datatype properties have a value range of data values linking instances to data values in the form of literals or XML Schema datatypes (Smith, 2003, 15).

(1) Restrictions on Properties. To define a property we must restrict what the property is relating or connecting to ensure it can be resolved to those classes. The OWL language accomplishes this in two ways, either by specifying the domain and range of the property, or by defining the property to be a specialization or sub-property of an existing property[77] (Smith *et al.*, 2003, 15).

(2) Specifying Domain. When specifying a domain we are asserting the domain values of this property must belong to the class extension of the class description. Such assertion is called establishing a domain axiom (van Harmelan et al, 2003, 26). Domains can be thought of as the class extension belonging to the node where an arc or property originates in terms of a directed graph. The OWL language also allows multiple domain axioms to be asserted. When this is done it should be treated as the intersection of the class descriptions the property relates (van Harmelan et al, 2003, 26). Simply put by asserting a domain we are restricting the set of classes eligible for a property to be assigned.

(3) Specifying Range. When specifying the range we are asserting the instances of a class are linked to either a class description or a data range. Such an assertion is called establishing a range axiom (van Harmelan, 2003, 27). Range can be thought of as a class extension or datatype belonging to the terminating node in terms of a directed graph. Simply put, by asserting a range axiom we are restricting the class extensions or datatypes eligible for a property to be assigned.

When we view the domain and range in terms of our example on a directed graph we can illustrate how a property is resolved to the respective classes by asserting our domain and range restrictions. With our example we chose to use the subClassOf property which is an embedded construct within the larger OWL construct[78]. Since subClassOf is embedded, the domain and range restrictions are encapsulated in the

[77] We will not cover subPropertyOf embedded property in this work. There are other formalizations within the OWL recommendation providing more advanced property restrictions we leave to the reader to study. The OWL Web Ontology Language Reference and the OWL Web Ontology Language Guide provide detailed explanation of allowable constructs. This work is concerned with showing enough of the basic constructs of OWL to show its value.

[78] OWL has 20 such embedded constructs (Patel-Schneider et al., 2003, 18-19).

logic abstracted by RDFS term subClassOf. Therefore, by asserting Dismounted Tactical Route is a subClassOf Tactical Route we have declared our domain and range restrictions automatically. In this case in order to be a Dismounted Tactical Route one must first be a Tactical Route. For illustration purposes only we will declare our domain and range explicitly for this example (See Figure 41).

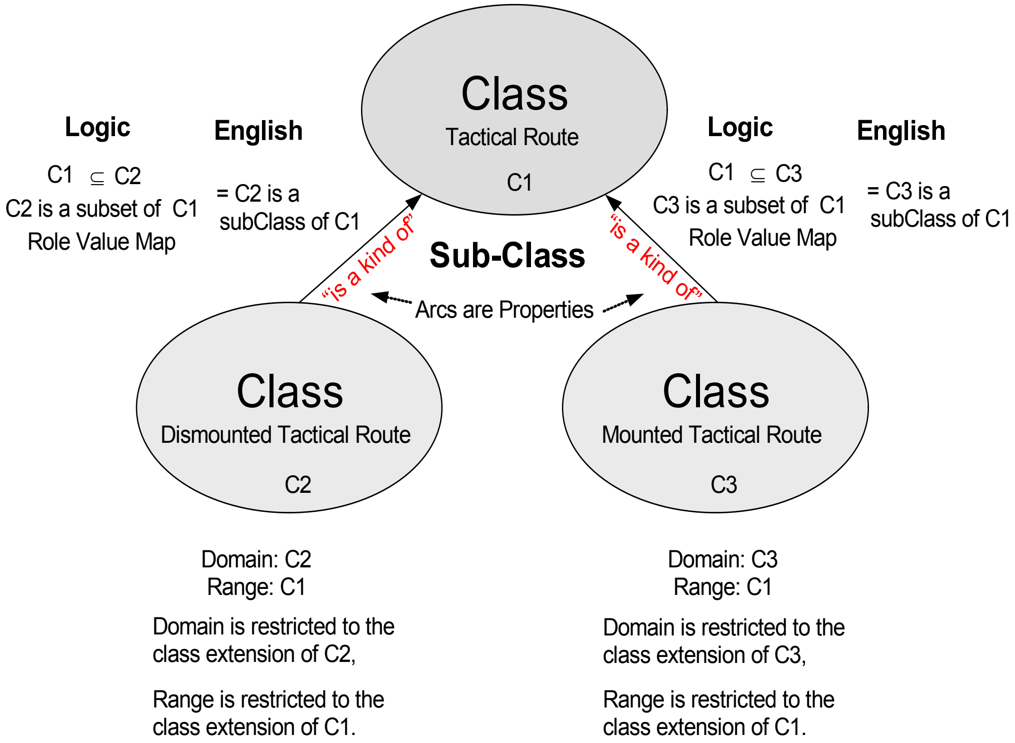


Figure 41. Properties.

To further examine our directed graph example notice that the arcs now represent the property. Listed along each of the arcs are the words "is a kind of." If an ontology developer wants to test the validity of a subclass, he should form a sentence with "is a kind of" between the two concepts and if it is true, then it is a subclass. For our example in Figure 41 our sentence would read, "Dismounted Tactical Route 'is a kind of' Tactical Route." Since this is true we have a valid subclass.

Another important detail to note about the Figure 41 is that it also depicts the exposure of the logic underlying the embedded concept subClassOf. It is important to reiterate the foundation of OWL is found in formal Description Logics. The syntax of OWL is a high-level abstract syntax with a model-theoretic semantics to provide formal, logical meaning (Patel-Schneider *et al*, 2003, 1)[79]. The abstract syntax is designed to make OWL easier to use and hide the complexity.

d. Instances

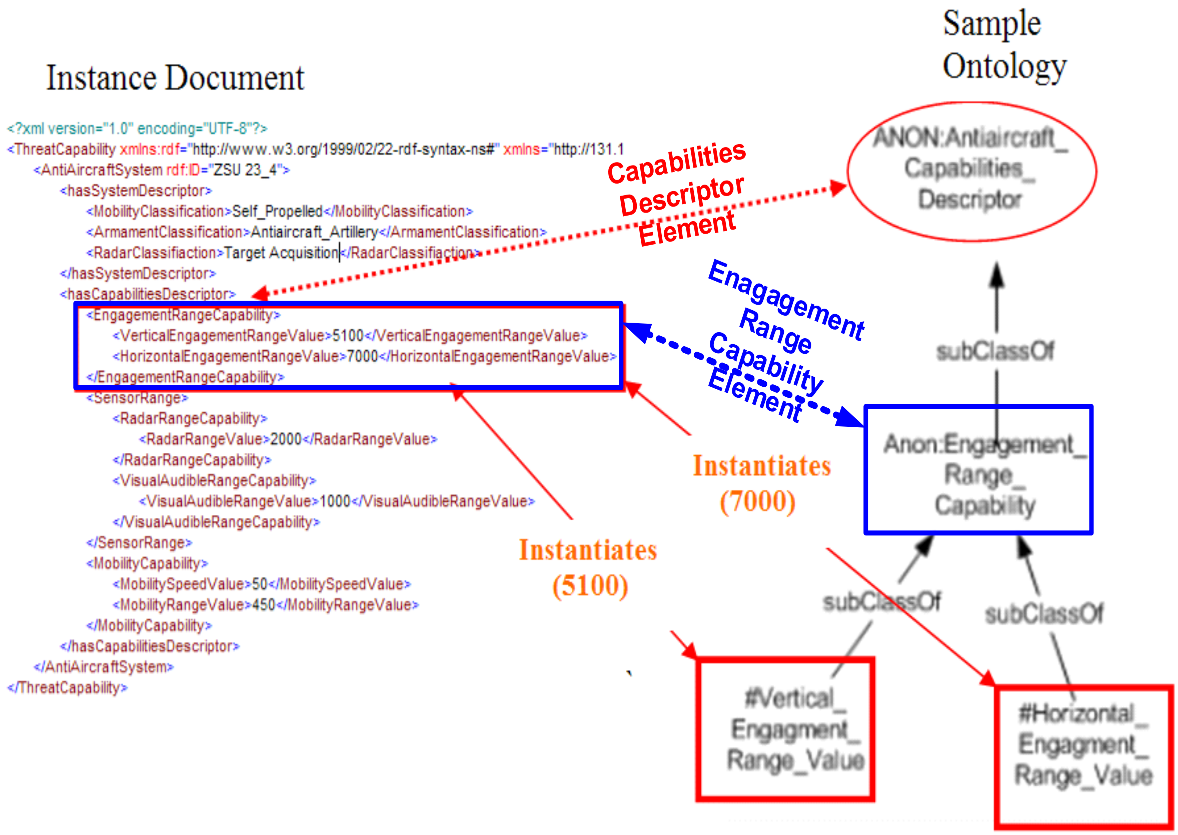
There are many who would argue instances or facts are not part of the basic elements of an ontology. Ontology Development 101: A Guide to Creating Your First Ontology by McGuinness and Noy consider instances as a step in ontology development, but when classifying elements of an ontology they treat instances as separate. McGuinness and Noy state an "Ontology together with a set of instances constitutes a knowledgebase (Noy, 2001, 3). Instances for the purpose of this work will also be treated as separate, but there are cases in which we will treat them as part of the ontology. Instances must be created from the constructs of the ontology; therefore, they cannot be treated totally separate and must be considered during design and implementation. It may be helpful to think in OO terms that instances of an ontology instantiate the ontology and the ontology, in return provides the containers for the instances to reside. The containers would be the class structure which also allows the instances to inherit meaning from the class descriptions. Therefore ontologies and instances have a symbiotic relationship. Now that we understand how we will treat instances in this work let us look at some formal definitions of what instances are and the roles they fill.

Instances are considered objects extensionally defined, or defined with regards to their existence based on a given ontology (Maedche, 2002, 63). Instances, also called individuals, are members of class extensions and are the relevant raw materials by which to extract intensional meaning (Maedche, 2002, 63). The OWL ontology provides the mechanism by which to describe the classes the instances belong and the properties they inherit by virtue of their class membership (Smith et al., 2003, 10).

At the document level an OWL instance is simply and XML document. The XML tag set is derived from the classes and properties that make up the ontology.

[79] OWL Ontology Language Semantics and Abstract Syntax contains two model-theoretic semantics. One is a standard model-theoretic semantics for OWL and the other is for RDF semantics.

Just as was stated in the previous section the instances bring the intensional meaning to the ontology. Figure 42 is a sample instance document and its corresponding Domain Concept Tree (DCT) from which the marked up, text based ontology was derived. Pay close attention to how the instance tag set is derived from the classes and properties of the ontology[80].


Instance Document
Sample Ontology
Capabilities Descriptor Element
Enagagement Range Capability Element
Instantiates (7000)
Instantiates (5100)
ANON:Antiaircraft_ Capabilities_ Descriptor
subClassOf
Anon:Engagement_ Range_ Capability
subClassOf
subClassOf
#Vertical_ Engagment_ Range_Value
#Horizontal_ Engagment_ Range_Value

```
<?xml version="1.0" encoding="UTF-8"?>
<ThreatCapability xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns="http://131.1
    <AntiAircraftSystem rdf:ID="ZSU 23_4">
        <hasSystemDescriptor>
            <MobilityClassification>Self_Propelled</MobilityClassification>
            <ArmamentClassification>Antiaircraft_Artillery</ArmamentClassification>
            <RadarClassifiaction>Target Acquisition</RadarClassifiaction>
        </hasSystemDescriptor>
        <hasCapabilitiesDescriptor>
            <EngagementRangeCapability>
                <VerticalEngagementRangeValue>5100</VerticalEngagementRangeValue>
                <HorizontalEngagementRangeValue>7000</HorizontalEngagementRangeValue>
            </EngagementRangeCapability>
            <SensorRange>
                <RadarRangeCapability>
                    <RadarRangeValue>2000</RadarRangeValue>
                </RadarRangeCapability>
                <VisualAudibleRangeCapability>
                    <VisualAudibleRangeValue>1000</VisualAudibleRangeValue>
                </VisualAudibleRangeCapability>
            </SensorRange>
            <MobilityCapability>
                <MobilitySpeedValue>50</MobilitySpeedValue>
                <MobilityRangeValue>450</MobilityRangeValue>
            </MobilityCapability>
        </hasCapabilitiesDescriptor>
    </AntiAircraftSystem>
</ThreatCapability>
```

Figure 42. Instantiation of an Ontology.

At the concept level an instance is simply a fact. A fact makes assertions about a domain of discourse. These assertions are considered true, until proven otherwise. Facts can be combined to inject both extensional and intensional meaning into an ontology. When this technique is used it can either be viewed as a knowledge base, by definition as we shall see in the Knowledge Base chapter, or a reinforced ontology. For

[80] The ZSU 23-4 instance is taken from a Threat Anti-Aircraft System Ontology which we cover later in depth.

our view we will adopt the latter. Even when seeded with a number of facts the ontology can only provide meaning based on facts asserted and described. Asserted and described facts are used to support additional fact assertions about other facts and assist with the discovery of implicit subsumptions[81] required by the application. In this case the pre-asserted facts are more intuitively aligned with the ontology discussion rather than waiting to include it in the knowledge base discussion.

e. Structure

A key aspect of an OWL ontology is its structure. Recall an ontology, in general, is a technique employed to represent knowledge about a domain of discourse. How we choose to organize the knowledge, or the classes which form the concepts that is the knowledge, becomes the structure of the ontology. Structure is simply a mechanism to visualize parts of a complex system and the relationships among those parts (Booch, 2001, 12) so humans may better understand it. In Chapter II we established knowledge as a complex concept (Obrst, 2003, 104) (Davenport, 2000, 9) (Reidl, 2002, 45). Complex concepts naturally are grouped into hierarchies (Booch, 2001, 9). Description Logics, the logical underpinnings of OWL, also rely on hierarchies to establish subsumptions (relationships between classes and subclasses) and classifications. From these facts about the organization of complex concepts and the criticality of hierarchies to DL, we can conclude that in order to be efficient, an OWL ontology must have an intrinsic, hierarchal organization[82].

To further support this concept we know OWL ontologies consist of classes and properties at the highest level of abstraction. Since there are levels of specialization implied by the degrees of abstraction, classes must be the entities forming the hierarchy of the ontology. In fact from “10,000 feet” one is able to discern a clear class hierarchy beginning at the top with the most general classes and ending at the bottom with the most specific subclasses. These classes are connected by properties that

[81] Implicit subsumptions will be covered in the Reasoning Section of the Knowledge Base chapter.

[82] There are some who design ontologies to contain more horizontal relations much like a Relational Database Entity Relationship Diagram. When designers of ontologies choose to design in this manner they lose some of the real power of ontologies such as inheritance, subsumption and multiple inheritance which the hierarchy provides.

may or may not be restricted with cardinality constraints. One should be able to visualize the instances instantiating the class extensions at the termination points of datatype or object properties. Hierarchical structure is a key enabling element of an OWL ontology.

The hierarchy embedded within an ontology is most visible by viewing the basic taxonomic structure or "skeleton" of the ontology reflecting only the concepts or classes. Figure 43 depicts the embedded hierarchies present within the Dismounted Ground Tactical Route Taxonomy. Notice how the level of specificity in the classes is more general in nature at the top of the hierarchy and increases as we near the bottom. Notice also how multiple hierarchies exist within a single ontology.

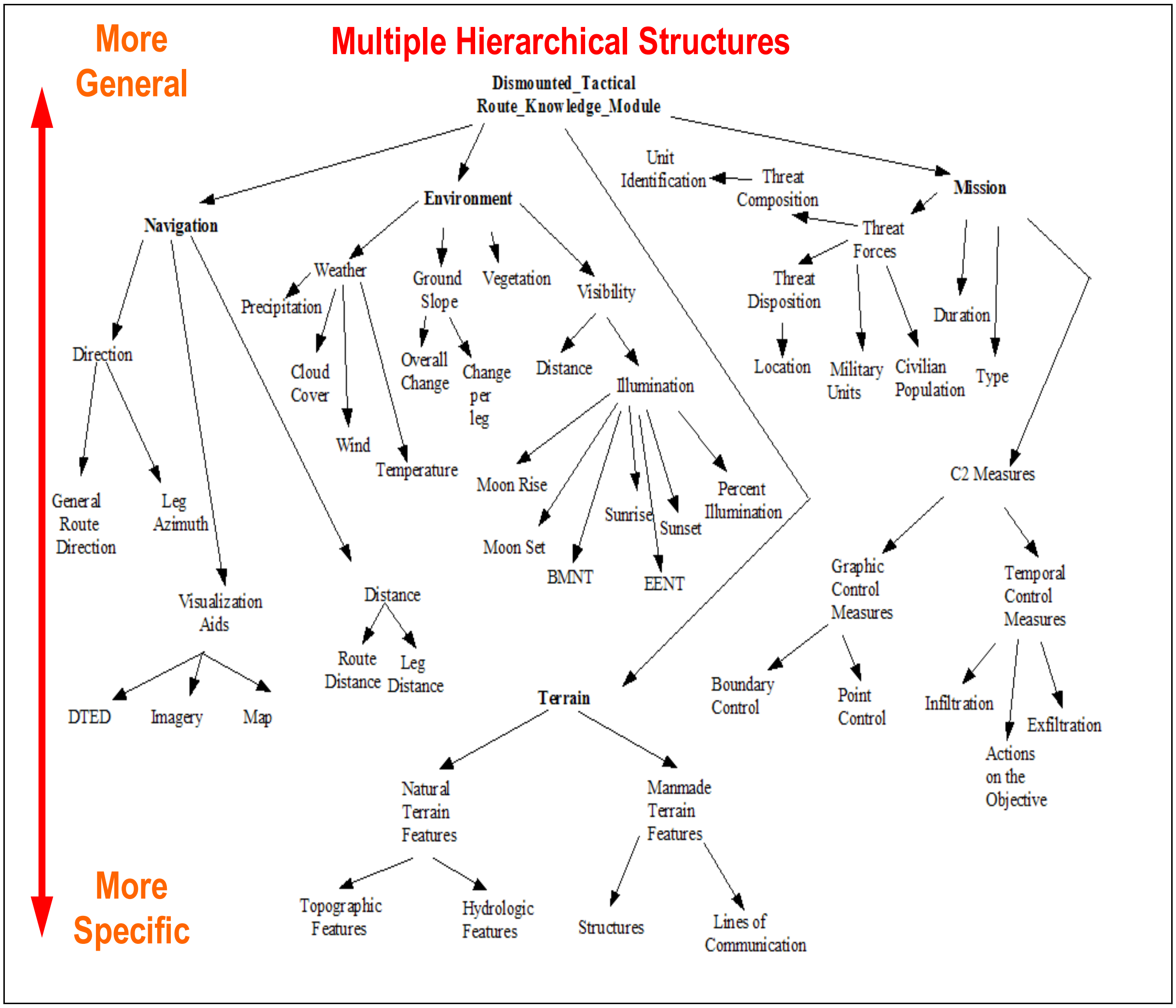


Figure 43. Embedded Taxonomic Structures within a Concept View of an Ontology.

D. ONTOLOGY DESIGN

1. Design Criteria

Thomas Gruber made the statement, “Formal ontologies are designed and, when we choose to represent something we are making design decisions” (Gruber, 1993, 2). So, before engaging in discussions involving specific patterns, techniques, and methodologies used to design effective ontologies, we must first review the general design decisions used to test, evaluate and guide us in the design process (Gruber, 1993, 2).

The design criteria we will analyze are consolidated from multiple sources, and what we present are the design criteria most applicable to military developers and users. Adherence to the general design criteria will help us avoid interoperability and reuse problems between ontologies.

a. Domain and Scope

In our definition of ontology we explicitly state that ontologies provide the meaning and context for a given *domain.* Therefore, to begin the ontology design process we must define our domain and establish the portion of the world we intend to model (Obrst, 2003, 127). Ontology developers must establish the likely boundaries and the scope of the target domain for the design to be effective. The boundaries of the domain may change during the design process, but it is often useful to establish a predicted domain from which to bound the initial scope of the design (Noy, 2001, 5).

It is important during our design process to make as few claims as possible about the domain allowing users and developers the opportunity to extend and instantiate the ontology as required (Gruber, 1993, 3). Problems relating to ontological commitment can be minimized by asserting the weakest theory (fewest claims) and still communicating the desired aspects of knowledge to drive the application (Gruber, 1993, 3). By attempting to contain all possible knowledge about the domain, a developer risks establishing too strong of a theory, limiting the flexibility of the ontology developer and user forcing them not to commit, or even consider using a given ontology (Noy, 2001, 19). Noy and McGuinness suggest the useful heuristic of only generalizing or specializing at most one extra level above and below the target granularity for the application (Noy, 2001, 19). Creating an ontology with as few claims as possible about the world is the designer's objective in order to maximize reuse and commitment. The bottom-line for a designer is not to represent any more in the ontology than what the application requires.

Let us apply this design principle to the ontology derived from the Dismounted Tactical Route Knowledge Module shown in Figure 43. First and foremost, we must establish the domain for this ontology. Unquestionably, our first impression indicates this ontology describes some concepts contained in the Military domain, as determined partially by the name, if we assume the designer used a descriptive naming convention. While some general members of the military domain will find this ontology useful, most will not since this ontology is targeted at a more specific user group of the military. Figure 44 below reveals the target users of this ontology to be the highly specialized group of Tactical Land Navigators. Tactical Land Navigators are the small unit leaders in Light Infantry or Special Operations Units. The target domain user would likely be able to relate to an ontology such as this, as it was written specifically for their use, in their specific language. It should be evident by bounding our domain our user group is more clearly revealed so the ontology can be specifically tailored to their system's requirements. In essence we have identified our expected user profile (McGuinness, 2000, 6) which still requires further investigation to uncover the specific purpose for which this ontology will be used.

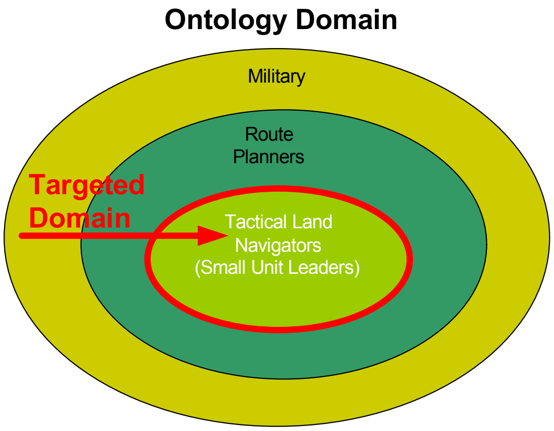


Figure 44. Domain Determination Example.

b. Purpose

Ontologies enable explicit, logic based representations of domain conceptualizations capable of being interpreted by both humans and machines. Ontologies should be designed for a specific purpose, usually to answer specific questions about a domain. Let us discuss the reasons an ontology might be implemented and some potential use cases within the military. We can use this preliminary discussion as necessary background that will assist us in our upcoming classification discussion.

2. Potential Military Uses of an Ontology

a. Command and Control

There are many potential uses for ontologies within the Military Domain similar to industry. With the development and maturity of the SWEB the ontology will be the cornerstone technology which shares a common understanding of a domain among humans, agents and machines. Ontologies for Command and Control Systems will be instrumental in establishing a Common Operational Picture (COP) among units by making domain analysis and assumptions explicit (Noy, 2001, 1). Agents assisting commanders with the Command and Control task have the ability to interpret data and know what in means based on the ontology.

b. Logistics

Ontologies can be applied to military logistics operations. An ontology could describe logistic processes in OWL-S and enable it as a SWEB Service. Not only can OWL-S describe simple atomic processes, but complex composite processes as well. A composite process is made up of more than one sub-process. The Ontology could include everything from reordering of ammunition to the procedures for coordinating its delivery on the battlefield. Service Description Languages like DAML-S and OWL-S are some of the newest ontology languages to be researched, but once realized will have great utility in this area.

c. Decision Support Systems

Some of the current applications of ontologies within the military we have seen have been in the Decision Support System niche. Ontologies supporting operations, logistics and intelligence can be greatly enhanced with implementations of effective ontologies designed to help speed a decision maker's net decision rate.

d. Modeling and Simulation

An important application area for ontologies is within the domain of Modeling and Simulation (M&S). Within M&S, an enormous amount of effort and resources are focused on development of the scenario or modeling environment. This task traditionally is painstakingly crafted for each scenario from a centralized location. A tremendous amount of domain expertise is poured into each scenario that are often times used and discarded due to the complex and time consuming nature of scenario modification. With the application of ontologies scenarios can now be pulled together from remote locations, data can be imported and the scenarios themselves can take advantage of the modular aspect of ontologies and modified with relative ease saving time and money.

Now that we have an idea of the potential employment areas for an ontology, let us look at Information Flow Framework (IFF) Foundational Ontology[83] and our abstracted derivative as a technique to provide a general classification to ontologies as a means to discover their general purpose.

Each of the IFF levels is designed to accomplish specific functions. Figure 45 shows the IFF Foundational Ontology Framework on the left, and an abstracted version derived from the IFF Foundational Ontology on the right. We will use the version on the right from which to establish our common Ontology Classification Framework to assist the military user and developer community in the formal classification of an ontology.

[83] IFF is the IEEE building block approach toward the development of an object level ontological structure [http://suo.ieee.org/IFF/version/20021205.htm].

e. IFF

At the most general level of classification the IFF asserts an ontology will either belong to the Meta Level or the Object Level. The Meta Level contains ontologies about ontologies (http://suo.ieee.org/IFF, 2002). One can liken this level to metadata in a database. It serves to describe the data, in effect it is data about data. In contrast the Object Level is where the domain content is described (http://suo.ieee.org/IFF, 2002).

Domain content can range from general to specific depending on the ontology's purpose. Figure 45 depicts a further decomposition of the Meta and Object Level Ontology categories as suggested by the IFF.

Within the Metalevel, ontologies are further decomposed by specialization. There is a Top Level supplying the general terms and relations. The Top Level is the most general category of ontologies. The next level, the Upper Level, provides the category theory, general classifications and concept analysis. Finally, the Lower Level provides the model theory and interfaces with the Object Level (http://suo.ieee.org/IFF, 2002). These stratifications provide useful generic applicability to gain insight about the contents and purpose of an ontology by its classification into one of these categories.

The Object Level can also be further decomposed as Figure 45 depicts. The decompositions to the categories of Generic, Middle and Specialized are an attempt to further classify ontologies within the Object Level by their degree of specialization. As one might assume by their names a Generic Ontology in the Object Level Classification would provide generic domain content, or content that could apply to a large number of applications across a domain. Following this logic an ontology classified as Specialized within the Object Level could be expected to provide very specific domain content to certain specialized segments of a domain. The Middle Level would contain ontologies not considered specialized, but specialized enough not to be considered generic. It is within this general classification framework we hope to assist users and developers in the military domain by associating function or purpose with a given ontology by its general classification category.

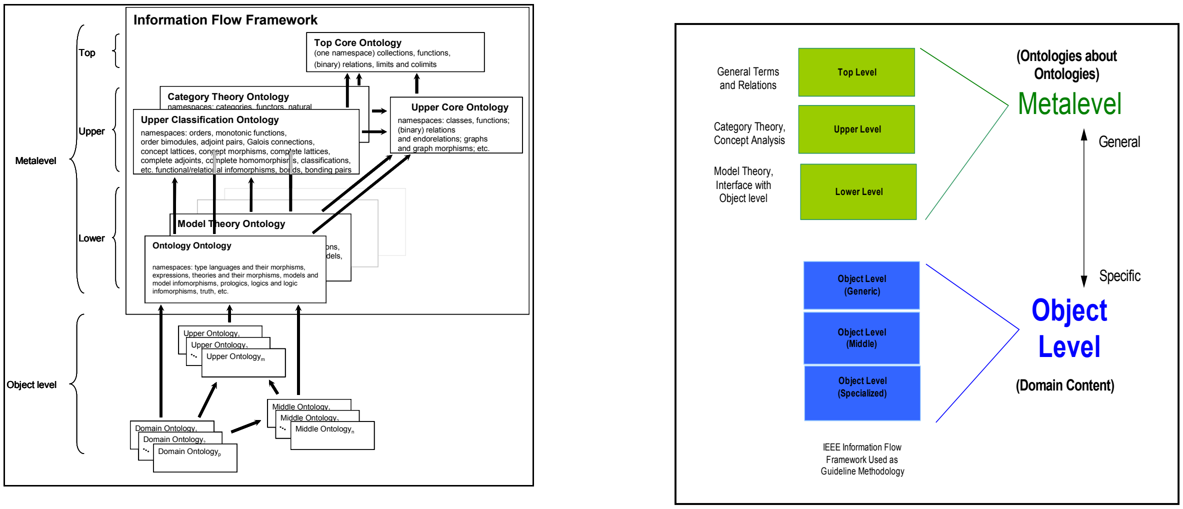

Information Flow Framework
Top
Upper
Lower
Metalevel
Object level
Top Core Ontology
(one namespace) collections, functions, (binary) relations, limits and colimits
Category Theory Ontology
namespaces: categories, functors, natural
Upper Classification Ontology
namespaces: orders, monotonic functions, order bimodules, adjoint pairs, Galois connections, concept lattices, concept morphisms, complete lattices, complete adjoints, complete homomorphisms, classifications, etc. functional/relational infomorphisms, bonds, bonding pairs
Upper Core Ontology
namespaces: classes, functions; (binary) relations and endorelations; graphs and graph morphisms; etc.
Model Theory Ontology
Ontology Ontology
namespaces: type languages and their morphisms, expressions, theories and their morphisms, models and model infomorphisms, prologics, logics and logic infomorphisms, truth, etc.
Upper Ontology1
Upper Ontology2
Upper Ontologym
Middle Ontology1
Middle Ontology2
Middle Ontologyn
Domain Ontology1
Domain Ontology2
Domain Ontologyp
General Terms and Relations
Category Theory, Concept Analysis
Model Theory, Interface with Object level
Top Level
Upper Level
Lower Level
(Ontologies about Ontologies)
Metalevel
General
Specific
Object Level (Generic)
Object Level (Middle)
Object Level (Specialized)
Object Level
(Domain Content)
IEEE Information Flow Framework Used as Guideline Methodology

Figure 45. IFF Foundation Ontology and Derived General Ontology Classification Framework (After: (http://suo.ieee.org/IFF).

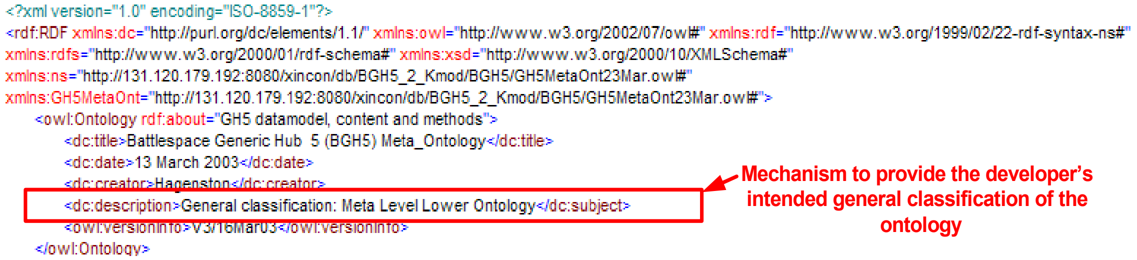
It is important to note there are many different methodologies proposed to classify ontologies, but no single standard has emerged[84]. We expect as the SWEB technologies continue to develop a standard classification framework will be necessary to differentiate and classify the different types of ontologies accessible on the SWEB. By implementing a general classification framework a potential adopter, whether it be an agent, service or human, would be able to readily identify the ontology's meeting the application's general requirement facilitating more effective discovery and higher potential reuse of existing ontologies. Reuse is an important design principle of the SWEB, as reuse will establish the connectedness required to achieve large scale common interpretation.

The general classification category for our example is "Meta Level Lower Ontology" and it could be inserted into ontology header in the Dublin Core Description[85] tag in the current OWL ontology constructs as we have done below. An annotated classification such as this would let potential users know the granularity level of the

[84] Jasper and Uschold have also developed a framework to classify ontology applications. They differentiated ontologies based on the application versus the function. We chose to discuss the functional view of classification in attempt to keep the application and the ontology as loosely coupled as possible. We expect this to be an area of continued research. We chose IEEE IFF Foundational Ontology as a basis for our classification framework due to its IEEE sponsorship and potential widespread adoption.

[85] Dublin Core is an online, interoperable meta data standard and specialized meta data vocabulary for describing resources that enable more intelligent information discovery systems (www.dublincore.org).

concepts and general content of the ontology without close inspection of the ontology[86]. If our general classification framework was adopted within our enterprise all who interpreted the dc:description tag would know the ontology's general classification and infer its intended purpose. Figure 46 illustrates general classification category in the Dublin Core tag set of our example ontology.



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
xmlns:ns="http://131.120.179.192:8080/xincon/db/BGH5_2_Kmod/BGH5/GH5MetaOnt23Mar.owl#"
xmlns:GH5MetaOnt="http://131.120.179.192:8080/xincon/db/BGH5_2_Kmod/BGH5/GH5MetaOnt23Mar.owl#">
    <owl:Ontology rdf:about="GH5 datamodel, content and methods">
        <dc:title>Battlespace Generic Hub  5 (BGH5) Meta_Ontology</dc:title>
        <dc:date>13 March 2003</dc:date>
        <dc:creator>Hagenston</dc:creator>
        <dc:description>General classification: Meta Level Lower Ontology</dc:subject>
        <owl:versionInfo>V3/16Mar03</owl:versionInfo>
    </owl:Ontology>
```

Mechanism to provide the developer's intended general classification of the ontology

Figure 46. General Classification in the Dublin Core Tag Set.

3. Evaluating a Specific Purpose with Competency Questions

In the last section we discussed potential uses of ontologies in the Military Domain and of the development of a standardized General Classification Framework. We are now ready to discuss the relationships between the specific purpose of an ontology and its competency questions. But before we continue, let us put the topics we discussed in the previous section to the test by employing them against the ontology we will be using as the subject of this section's discussion.

For the purposes of this discussion we will introduce an ontology developed to coordinate interoperability between an external data source and an internal knowledge base. We chose to solve this problem by describing the external data source's data model, content, and methods in a manner enabling both humans and agents to interact with the external source. The subject of our focus was the Generic Hub 5 NATO Data Exchange Database referred to as Generic Hub 5 (GH5). GH5 is designed to assist the nations of NATO exchange militarily significant data and establish common meaning for terms. Let us examine Figure 47, the Domain Concept Tree for the GH5 ontology. The ontology we developed conceptualizes the meta data aspect of the GH5

[86] Assuming the classification was done correctly in accordance with some standard and that it was done without the intention to deceive.

system and exposes it in the form of an ontology for agents/systems to gain an understanding regarding the data model, content, methods and articulation mechanisms. Our ontology serves as an interoperability entry point for GH5.

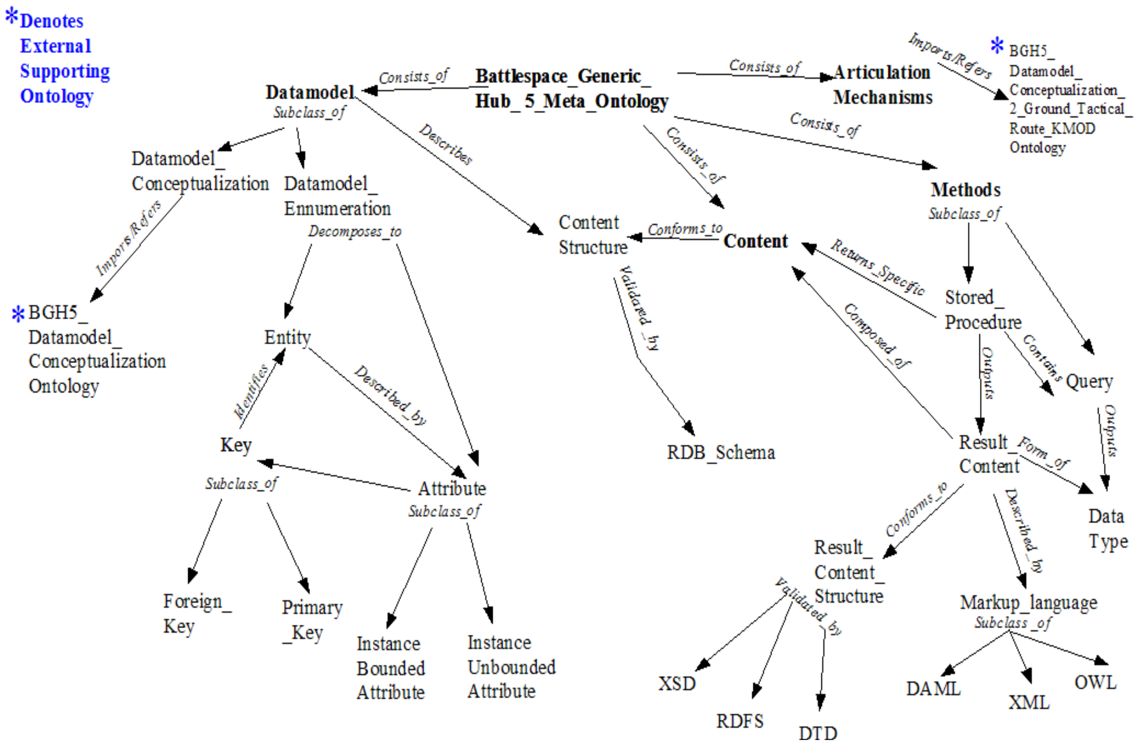


Figure 47. Specific Purpose Example.

As we stated above, we will apply the General Classification Framework we derived to our example ontology. We have already revealed the solution, but we can still add to the understanding of the idea behind general classification by applying the classification framework to our example ontology.

First, analyze the name of the ontology. As with the OO design methodology one can draw many inferences about an ontology's purpose from its name. This ontology is named Battlespace Generic Hub Meta Ontology. If the designer of this ontology used some general naming conventions we can derive enough information from the name to assist us in our classification effort. Defining naming conventions for ontologies and concepts within ontologies and strictly adhering to them helps make an ontology relatively easy to classify (Noy, 2001, 21). The "Meta" in our example's name prompts us to make a possible association with the Meta Level of the classification framework. We know according to the framework that Meta Level ontologies are ontologies about other ontologies. This is the first concrete criterion for associating an ontology at Meta Level of our general classification framework. Our example ontology thus far, looks to be a potential member of this Meta Level category.

From Figure 46, we see our ontology imports two other ontologies. According to their names, one of them is a conceptualization of a data model and the other is an articulation ontology. Since the BGH5 Meta Ontology contains at least some information about two other ontologies (location and some degree of content) we can safely assert our example ontology belongs to the Meta Level category.

Determining the sublevel is slightly trickier and requires close analysis of the ontology's content and structure. For our example ontology we get the impression from the number of ontologies it imports and the type of details it contains that it is likely candidate for an interface with Object Level. We can arrive at this almost entirely by the process of elimination. We can ascertain this ontology is more specific than we would expect a Top or Upper Level ontology to be. We would expect more general concepts lacking specificity. The class structure within our ontology has details that seem to relate to other more detailed concepts likely residing within the Object Level. Just as a refresher, the Object Level contains the domain specific knowledge. Because of this we can assert it is a candidate for membership in the Lower Category of the Meta Level. While this classification was rough shot, it was intended to demonstrate by the name, structure and granularity of the content alone one can generally classify an ontology within the simple framework we established relatively quickly and without in depth analysis. It is our strong position a general classification should require little analysis and be done quickly.

To demonstrate the role of competency questions in evaluating an ontology for adherence to a specific purpose lets examine the competency questions associated with our GH5 Meta Data Ontology:

- How is the data stored in the GH5 relational database?
- What stored procedures or queries are available for use?
- What is the structure of the query outputs?

As we can see from the competency questions above all are related to our overarching purpose of interoperability with the GH5 database. Specifically, how were the data elements stored, how can I access them (queries) and what is the structure of the output of the query. Providing the ontology delivered this information, we could achieve the interoperability we required with data source. To reinforce this point we will examine one of the competency questions in detail to demonstrate how the ontology provided the information we required.

4. Competency Question Analysis: How Is the Data Stored in the GH5 Relational Database?

Interoperability between the data source and the knowledge base was the overarching goal. To achieve a level of interoperability we established a data model class in our Meta Ontology with two subclasses, data model enumeration and data model conceptualization. The data model enumeration decomposed to a series of subclasses representing the atomic elements of a relational data model and their relationships (See Figure 48). From the data model enumeration class a human or an agent could reasonably determine *what* the Entities and Attributes where for the model, and determine the highly specialized Attributes serving as Primary and Foreign Keys. This portion of the ontology provided a listing and function of the data model's elements, but it failed to describe the specific relations between a given entity and its attributes including which of its attributes were the Primary or Foreign keys.

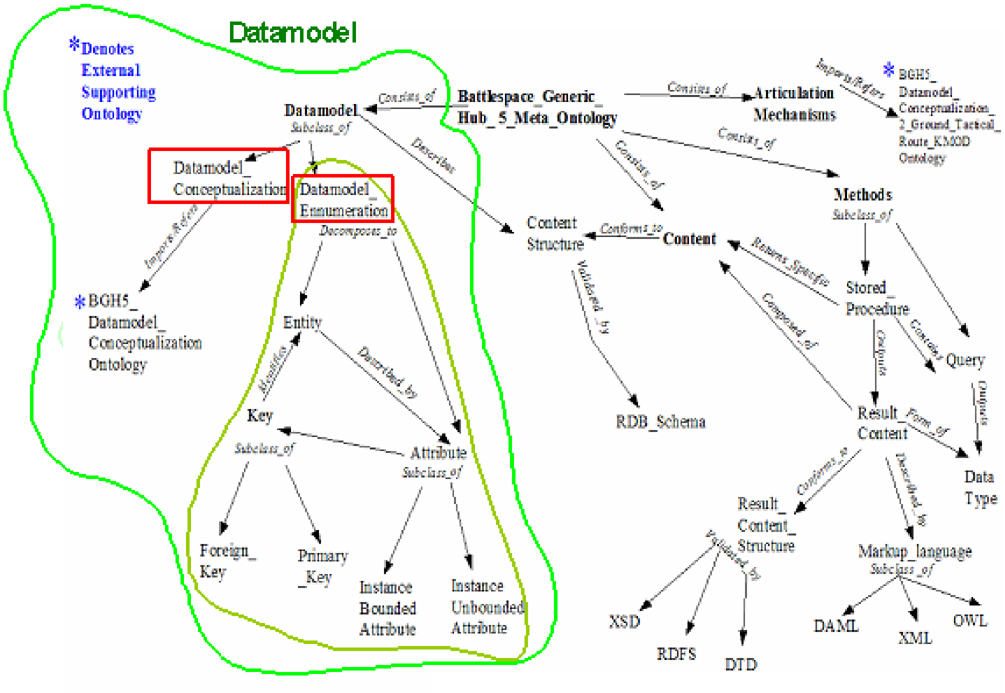


Figure 48. Data Model Enumeration Subclass.

To expose the specific relations of each entity we developed another ontology conceptualizing the data model itself (See Figure 49). The Data model Conceptualization Ontology replicated the role of the Entity Relationship (ER) Model[87] and added additional expressiveness. The Data model Enumeration Ontology used in conjunction with the Data model Conceptualization Ontology adequately assisted in providing more complete answers to our competency questions we established above. The more completely we are able to answer our competency questions the more effective the ontology.

[87] The future of the Semantic Web may allow the ability to "inject" meaning into objects like ER diagrams without creating an additional ontology. Related, there are several studies ongoing with semantically enabling the Unified Modeling Language (UML) which would also obviate the requirement for an additional ontology.

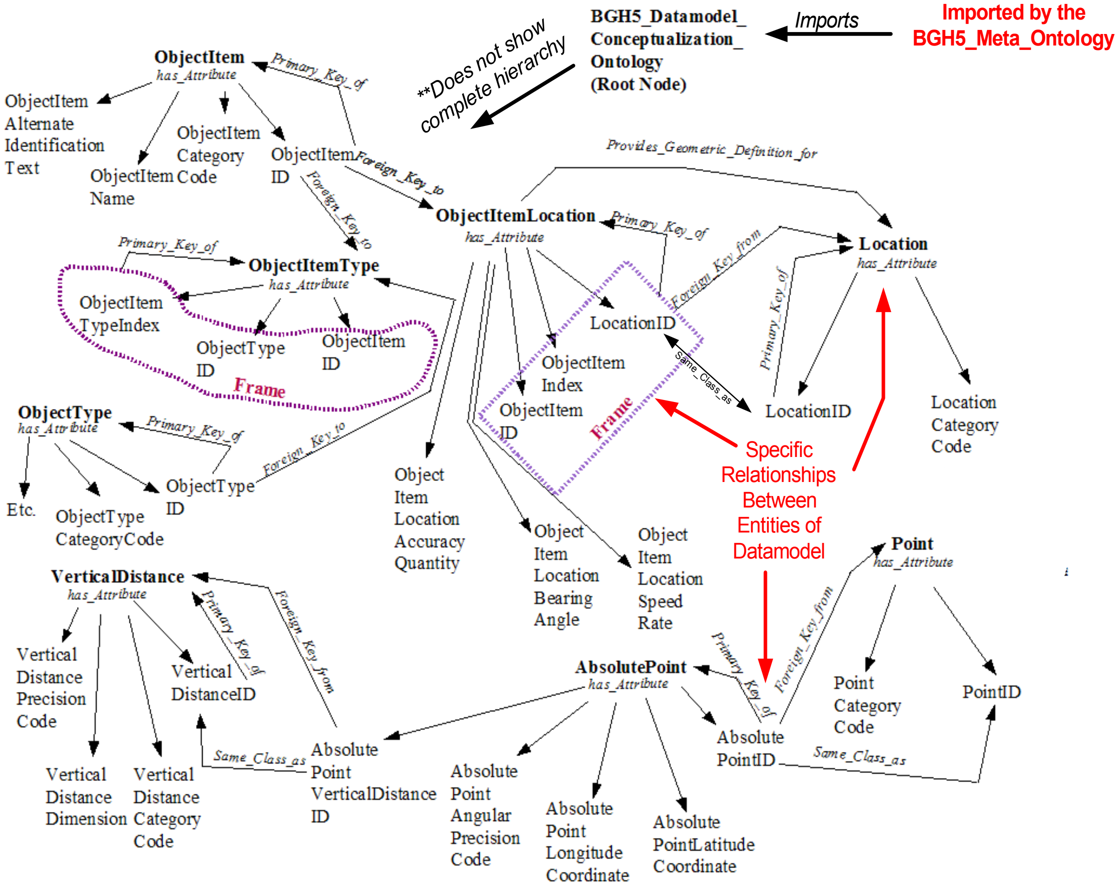


Figure 49. Data Model Conceptualization Domain Concept Tree.

5. Extendibility/Reuse

Designing for extendibility and reuse are critical concepts to the realization of the SWEB. In order for the SWEB to reach its full potential and attain the lofty title of "web of meaning or web of semantics," the multitude of separate ontologies must be linked and related to each other ([2] Klein, 1) and used in combination with other ontologies (Kim, 2002, 2). It is postulated that numerous, locally consistent, but globally heterogeneous ontologies will exist with no central ontology aware of these local ontologies (Kim, 2002, 3). But, each of the local ontologies will be aware of their neighbors and a few will have links to more distant ontologies weaving together small interconnected worlds. Multiple copies of ontologies can exist to mitigate the effects of network outages and malfunctioning URLs. The concept of a local cache discussed in the Network Data Sources Chapter not only applies to data sources, but to knowledge sources as well. An ontology serving a critical function in an application must be mirrored locally to ensure access and functionality when the network is unavailable. To ensure the full power of ontologies are realized ontology designers must make maximum use of existing ontologies on the network by either extending or reusing them. Both options can help achieve the small world effect in slightly different ways. Let us first examine the idea of reuse.

Ontology reuse implies the utilization of a preexisting ontology without modification to its original state or location. Reuse is fundamentally dependent on shared conceptual foundations for the domain of discourse it describes (Gruber, 1993, 3). To simplify, the ontology being considered for reuse must describe objects of the world in a way the implementing application can commit to. If agreement cannot be reached the designer continues to search for a different ontology to reuse in line with the application's conceptual requirements or design a new one. Reusing existing ontologies may be a requirement in order to interact with other applications that may already be committed to other ontologies or controlled vocabularies (Noy, 2001, 6). Designing a new one independent of existing ontologies does little to further the goal of the "web of semantics" unless other ontologies link to it after it is deployed. Critical to the "web of meaning" is this idea of reuse.

Ontology extendibility implies the specialization of an existing ontology (Gruber, 1993, 3) for use in a different domain or problem space than what it was originally designed. As implied by specialization, the ontology being extended is usually more general than the target application requires. Ontologies are extended by simply adding additional constraints or relationships on the existing ontology. The result is a reuse of concepts.

A central idea to the reusability and extendibility of ontologies is the ability of the designer to anticipate additional usages for the vocabulary and concepts that will be contained within the ontology. An ontology designer has little influence on the concepts introduced by the ontology, as they will usually be driven by the requirements of the application, but there is considerably more influence regarding the vocabulary. Gruber states "One should be able to define new terms for special uses based on the existing vocabulary, in ways that do not require revisions of the existing definitions (Gruber, 1993, 3).” In other words controlled vocabularies should be used whenever possible and specialized as required to meet the needs of the application. With this in mind let us examine controlled vocabularies

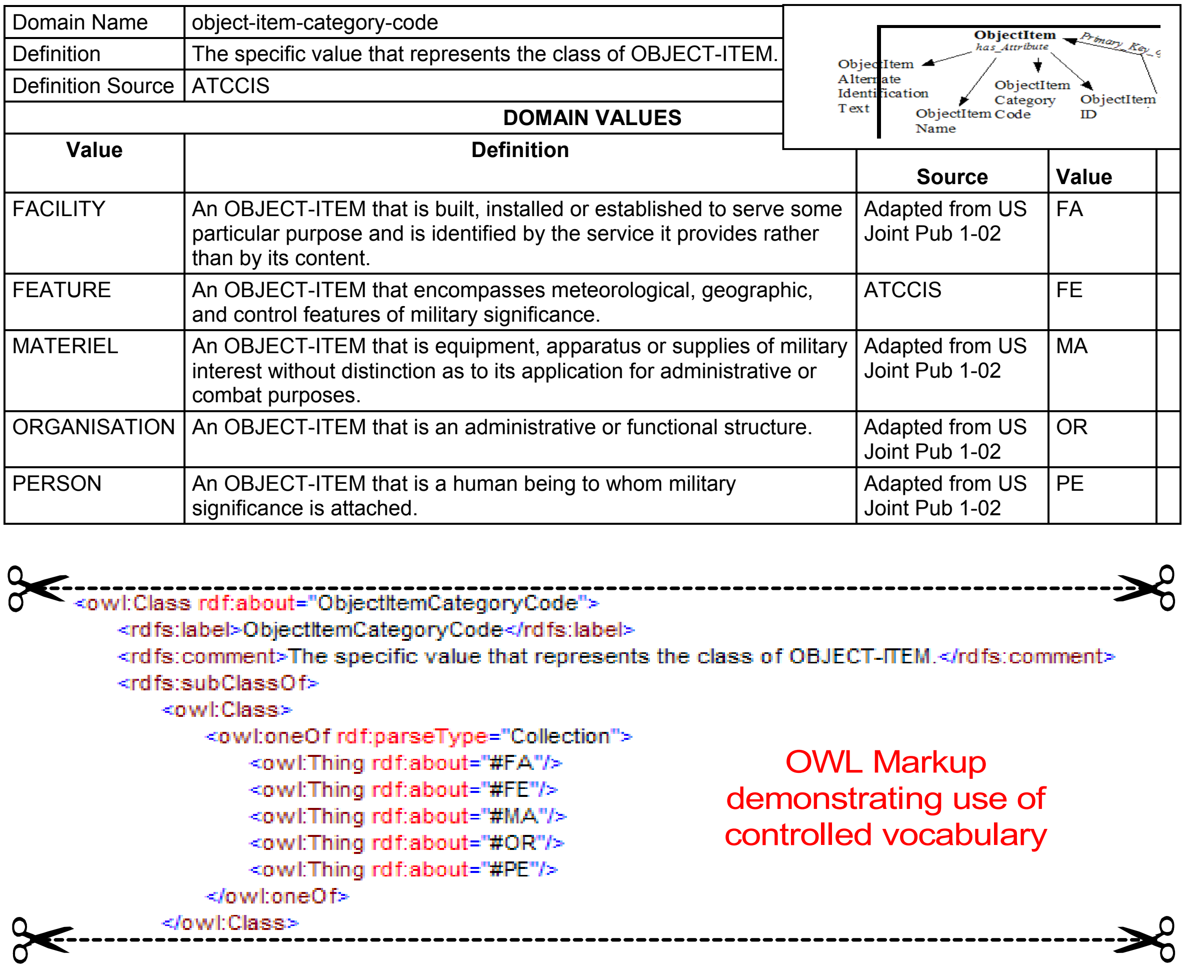
The concept of controlled vocabularies is an important discussion point to designing ontologies and establishing the SWEB. The adherence to a controlled vocabulary can be informal or formal. Informal adherence occurs naturally and persists because the members of a domain wish to communicate. When members introduce new terms into a domain the terms are either accepted by the domain or rejected. Ultimately individuals become attuned to each other’s terms and concepts, otherwise communications break down (Gardenfors, 2000, 155). The military has many occurrences of informal adherence to vocabulary within its ranks. If one were to examine various groups within the military one would immediately notice the presence of an informal controlled vocabulary. This informal vocabulary is the mechanism its members use to communicate within their group. No policy or authority directs the members to communicate using this language, it is not captured in any book, and it just emerges predictably across the group.

Formal adherence to a controlled vocabulary is directed by doctrine, policy, or specification. Both allow for common exchange of terms amongst committed parties, but formal adherence guarantees members of domain are committed to the vocabulary. According to McGuinness, “As ontologies become more common within applications and those applications become larger and longer lived, it is becoming increasingly common for ontologies to be developed in distributed environments by authors with disparate backgrounds (McGuinness, 2000, 1). A formally controlled language will assist in grounding the concepts with common definitions and terms especially in distributed environments. Davenport and Prusak summed this problem up by stating, “People cannot share knowledge if they do not speak the same language” (Davenport, 1998, 98).

Within the military, and the Army specifically, all operations and actions conducted are based on doctrine found in a training or field manual. As we mentioned above there is argument about how closely doctrine is followed in practice, but the terms and concepts found within the doctrine are widely adopted across the service. This could be the grounding required for formal adherence, although at this point nothing directs members of a domain to communicate in doctrinal terms; it very much happens informally. Even so, a small number of terms and concepts in wide use could become the foundation of a controlled vocabulary based on those doctrinal terms and concepts. To press forward with this idea to establish a controlled vocabulary the doctrine writers would become the custodian and authority of the terms. To use a term, the namespace or Published Subject Indicator (PSI)[88] (Vatant, 2003, 74), an artifact from the XML Topic Map (XTM) community, could be included as a reference namespace grounding the term within the conceptual foundation of the originating doctrine. PSIs within topic maps are actual binding points for subject identity (Vatan, 2003, 74) and they could be used similarly within an ontology. Once the term is grounded the meaning cannot be changed unless done so by the doctrine writers. This guarantees conceptual grounding, a common vocabulary likely to enjoy higher instances of extendibility and reuse among its published ontologies.

For a practical example of implementing an ontology with formal adherence to a controlled vocabulary, we can refer to the GH5 Data Model Conceptualization Ontology. GH5's Controlled vocabulary was used exclusively in the design of this ontology. The GH5 Specification enumerates in exhaustive detail the definition, pedigree and allowable physical values for each term. Figure 50 below, depicts the Generic Hub 5s controlled vocabulary specification for a single attribute in the data model. We can conclude from Figure 50, if this vocabulary is a representative sample, controlled vocabularies can be extensive and should be used whenever possible to ensure common definitions of terms. The biggest benefit however, is someone else did the work.

[88] The idea of PSI is not unique to the XTM community. Within the Unified Medical Language System (UMLS) they apply this grounding to not only terms but concepts. They use a Concept Unique Identifier (CUI). The CUI underlying concept is based on Ogden and Richards meaning triangle we introduced in the Knowledge Chapter.



Domain Name	object-item-category-code			
Definition	The specific value that represents the class of OBJECT-ITEM.			
Definition Source	ATCCIS			
	DOMAIN VALUES			
Value	**Definition**	**Source**	**Value**	
FACILITY	An OBJECT-ITEM that is built, installed or established to serve some particular purpose and is identified by the service it provides rather than by its content.	Adapted from US Joint Pub 1-02	FA	
FEATURE	An OBJECT-ITEM that encompasses meteorological, geographic, and control features of military significance.	ATCCIS	FE	
MATERIEL	An OBJECT-ITEM that is equipment, apparatus or supplies of military interest without distinction as to its application for administrative or combat purposes.	Adapted from US Joint Pub 1-02	MA	
ORGANISATION	An OBJECT-ITEM that is an administrative or functional structure.	Adapted from US Joint Pub 1-02	OR	
PERSON	An OBJECT-ITEM that is a human being to whom military significance is attached.	Adapted from US Joint Pub 1-02	PE	

```
<owl:Class rdf:about="ObjectItemCategoryCode">
    <rdfs:label>ObjectItemCategoryCode</rdfs:label>
    <rdfs:comment>The specific value that represents the class of OBJECT-ITEM.</rdfs:comment>
    <rdfs:subClassOf>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="#FA"/>
                <owl:Thing rdf:about="#FE"/>
                <owl:Thing rdf:about="#MA"/>
                <owl:Thing rdf:about="#OR"/>
                <owl:Thing rdf:about="#PE"/>
            </owl:oneOf>
        </owl:Class>
```

OWL Markup demonstrating use of controlled vocabulary

Figure 50. Generic Hub 5 Controlled Vocabulary and OWL Markup Example.

6. Ontology Design Methodology Highlights

Thus far we have discussed purpose, scope, reuse and extendibility. We have belabored the use of restricted vocabularies whenever possible, gaining commitment and imposing minimal detail to achieve the ontology with the weakest theory and most meaning. Now we need a methodology to capture these heuristics to ensure we avoid the design pitfalls we discussed above. An appropriate design methodology offers us a useful mechanism to accomplish this. Our research lead us to many different methodologies. Many were a few paragraphs outlining simply "how to build" an ontology. Others were lengthy and detail focused. For our purposes we focused on the best aspects of the methodologies we researched to assist a military user to evaluate various methodologies for implementation. We chose to highlight points specific enough to help an adopter seeking a methodology find the appropriate one. Each of our highlights will be traced to their parent methodology and end with a value added analysis. Where applicable we applied other design principles from earlier discussion topics to reinforce important points.

a. Feasibility Assessment (FA)

Most of the methodologies we reviewed assume a feasible application area has been determined and the conditions are right for implementation of an ontology. Only one of the methodologies we researched specifically mentioned a requirements specification. Explicit generation of initial requirements is what the FA accomplishes and is why the FA must be included. In fact only the On-To-Knowledge Methodology (OTK) (Sure, 2003, 34), adapted from CommonKADs[89], included an FA. We view the FA as serving a necessary requirements analysis function early in the development process to drive design. Additionally, the value of establishing a baseline, structured approach early in ontology development will improve chances of a attaining a requirements based final design. The FA serves as a pre-domain study focusing on quickly identifying facts, assumptions, goals, constraints, and establishing initial requirements. This phase is not intended to be exhaustive in its conclusions, but to provide a start point to the formal design phase. The following checklist includes key concept areas the FA should focus are:

- Analysis of the preliminary usage scenario/application
- Establish initial requirements
- Estimate who users will be
- Estimate of domain
- Estimate of scope
- Identify requirements for domain experts
- Recommendations for Controlled Vocabularies
- Identify existing ontologies in the same or related problem space
- Preliminary choice of language

[89] CommonKADS is a methodology to support structured knowledge engineering using clear links to OO development techniques compatible with UML (www.commonkads.uva.nl/frameset-commonlads.html)

- Preliminary selection of automated tools
- Identify ontology development team

In addition, the following pre-design competency questions (Obrst, 2003, 127) are helpful to consider:

- What is the ontology intended to be about (in general)?
- How will the ontology be used? (preliminary estimate)
- What do you want to state in the ontology (preliminary estimate)?
- What modifications will be required over time?

Armed with estimates, if not answers to these questions, enough information is available to form a decision as to whether ontology development is feasible and should continue. The FA is iterative and can be executed as many times as necessary and be as thorough as time allows. The OTK and CommonKADS Ontology methodologies both include the FA in their methodology.

b. Enumeration of Terms, Concepts and Relationships

An organized brainstorming process to capture meaningful concepts, terms, relationships right down to the subjects, verbs, objects and adjectives is critical in capturing the objects of a domain (Obrst, 2003, 127). Most advocate capturing a list of terms we would like to make statements about or explain to users (Noy, 2001, 6) (Obrst, 2003, 127). Others advocate describing the concepts and then associating terms (Sure, 2003, 43). No matter how this is accomplished the end state must be a list of terms, concepts and how they relate evaluated against the scope and purpose. OTK advocates focusing on the most important concepts and then through generalization or specification identify the remainder of the concepts (Sure, 2003, 43). This technique is called a Middle-Out approach. The same technique to discover potential concepts can be done as effectively from the Bottom-Up or Top-Down. As long as the concepts are captured along with their definitions and potential usage of controlled vocabularies are considered how this step is done is arbitrary.

c. Form the Class Hierarchy

When the concepts and terms are enumerated next we must relate the entities in our universe of discourse by developing the class hierarchy. McGuinness and

Noy state there is no single correct hierarchy for a given domain (Noy, 2001, 12). However, the class hierarchy is what makes inheritance and subsumption possible within an ontology. The inheritance enabled by the hierarchy within the ontology is responsible for inference engines being able to reason over a hierarchy with predictable results. One such inference operation dependent on the hierarchy is the operation of classification. Classification amounts to using the hierarchy to place a new concept in the appropriate place in the hierarchy and checking subsumption between each defined concept and the new concept to ensure the placement is valid (Nardi, 2003, 14). Without a correct hierarchy this operation would likely achieve unpredictable or unsatisfiable results. To avoid this problem, the class hierarchy must be planned carefully and checked to ensure the logic is correct. Yes, it is true there is not one right hierarchy for a given domain or ontology, but the hierarchy can be logically wrong. The class hierarchy must be driven by the requirements of the application and care should be taken to ensure it is logically correct.

To ensure the hierarchy is correct, check the subordinate class to see if it is related to the more general class by an "is-a" relation (Noy, 2001, 12) (See Figure 51). For instance, if the class Point Control Measure is the most general class, then all of its children, the members of the more specialized classes, should be a "kind of" Point Control Measure. If we say Way Point is one of those specialized classes then we should be able to put it to the "is a" test. The 'is a" test reads, "A Way Point is a Point Control Measure." If this is correct then the hierarchy is also correct.

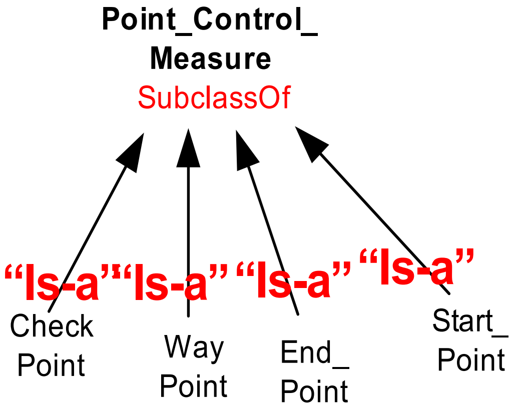


Figure 51. "Is –a" Test.

To further assist with ensuring the correctness of a hierarchy a designer can use visualization techniques. Domain Concept Trees, such as the ones we have used throughout this work to better visualize the example ontologies are an effective and simple tool to both graphically depict the ontology, as well as expose the hierarchy. Additional uses of Domain Concept Trees include relating classes with properties, spotting inconsistencies within an ontology, and as an effective pseudo code replacement while marking up an ontology into its computational model (Fernandez, 8). Besides Domain Concept Trees there is also a more specialized version of the same technique called Attribute Classification Tree. The Attribute Classification Tree is used to graphically depict attributes and their inference output.

7. Design Patterns

A pattern is defined as an idea that has been useful in one practical context and will probably be useful in others (Fowler, 1997, 8). Patterns provide a starting point from which to leverage practical usage of a concept and apply it to a new one. While undoubtedly there are many useful patterns that exist in the ontology design world, two especially useful patterns emerged during our research are worthy of discussion. The first is the concept of an articulation ontology, and the second is the concept of a highly specialized, multi-disciplined ontology called a knowledge module. Both patterns will be described in detail.

a. Articulation Ontology

As we stated above, for ontologies to have the maximum impact and reach their full potential they must be widely shared and reused ([2] Klein, 1) (Kim, 2002, 2) (Smith et al, 2003, 24). Sharing and reusing ontologies is done by merging, combining or articulating between terms and concepts in a given ontology, to terms and concepts in another. This process is unfortunately not as simple and arbitrary as mapping from one ontology to another[90]. The underlying semantics and relationships associated with the

[90] 1 to 1 mapping can work if the conditions are right.

ontology must also be mapped to preserve the underlying computational model. To accomplish this, an articulation[91] must occur on multiple levels (Klein, 2). Let us examine some of the details and patterns involving this concept.

The OWL Language Guide asserts, "Much of the effort of developing an ontology is devoted to hooking together classes and properties in ways that maximize implications[92] (Smith, 2003, 24). The goal of OWL is to create the environment where simple assertions about class membership have broad and useful implications for the SWEB (Smith, 2003, 24). These "broad and useful" implications translate to almost automatic reuse and integration potential between ontologies and applications on the SWEB. The OWL language construct allows a user the ability to easily import external ontologies into the current ontology, but currently provides no mechanisms to deconflict the potential consequences of the import. McGuinness states, "Merging small ontologies may not be difficult to do manually, but once the ontologies become large, it becomes more critical to provide systematic tool support ([2] McGuinness, 2000, 9). Today the task of combining, merging and articulating between ontologies is largely a manual endeavor[93].

To begin let us describe and examine the specific mismatches that must be overcome to attain the goal of reuse and sharing of ontologies on the web. Mismatches between ontologies can occur at two levels, the language/syntax level or the ontology/model level (Klein, 2).

(1) Language Mismatch. Language level mismatches occur when ontologies are written in different language constructs (Klein, 2). These mismatches are often the easiest to detect and can be remedied by rewriting the concept from one ontology into the desired markup language. This type of mismatch however can often suggest there are deeper mismatches within the model as different knowledge representation languages often express logical constructs differently. This type of

[91] Articulation or Translation is defined as changing the representation of the formalism of an ontology while preserving the semantics (Klein, 2).

[92] Implication is defined as a logical relation between classes and/or propositions.

[93] There are a few automated/semi automated applications capable of assisting with these tasks: ECIMF Semantic Translation Tool, Chimaera, Buster, OntoMerge, SHOE.***List is not exhaustive.

language level mismatch will likely be mitigated by the adoption of OWL as the W3C Ontology Language Recommendation. We will likely see, as has been the case with DAML, users of legacy languages migrating to OWL out of interoperability necessity. This natural attuning of the language constructs should provide the required focus for tool makers to design tools with a better chance of widespread adoption to put the onus of mismatch detection on the software tool. Other types of mismatches will not be alleviated by the adoption of OWL and are found at the ontology or model level

(2) Model Mismatch. A mismatch at the model level can occur when two ontologies model objects by imposing varying levels of detail (Klein, 3). We will call this mismatch one of physical granularity as it relates to the description of the physical characteristics of the object being modeled. Before merging or combining ontologies the level of physical granularity must be examined to ensure a match.

An example of a physical granularity mismatch can be demonstrated with the example ontologies shown in Figure 52 regarding Antiaircraft Radars. Both ontologies are addressing the characteristics of the systems radar range, but one uses specific integer values and other uses a relative English description. Attempting to map the value of 8200 from one ontology to Medium Range in the other ontology would not give us the desired results. To set up this example we must make the assertion Medium Range is between 5000-10000 meters and that our value of 8200 is within the Medium Range category. With that said, the problems associated with this mapping would now treat 8200 as a Radar Range Classification in one ontology and a non-negative integer associated with the has_RangeValue data type property in the other. The has_RangeValue data type property also carries with it restrictions on its domain and range not preserved in the mapping. As is evident, this direct mapping will create a conflict within the ontology and deliver undesirable results.

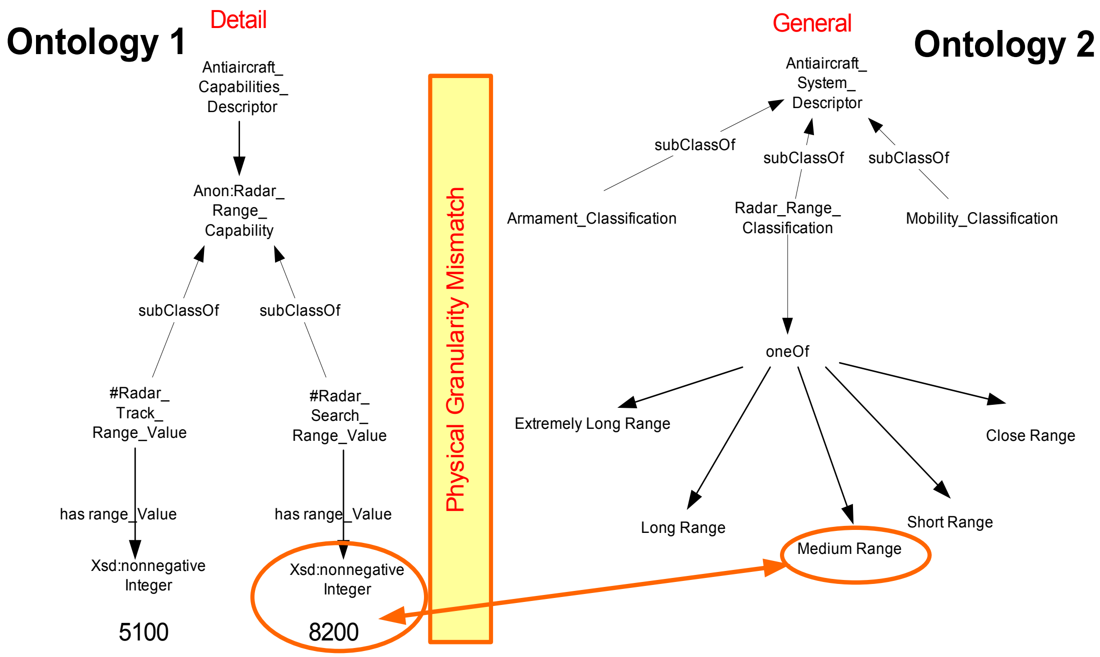


Figure 52. Physical Granularity Mismatch Example.

Granularity not only exists on the physical level, but it exists on the conceptual level as well. If one ontology conceptualizes its objects from a physical characteristic perspective and the other from a data model perspective, a mismatch has a great potential to occur. This conceptual granularity mismatch can occur even if the terms are what we would consider intuitively close in meaning. As an example let us take two ontologies both modeling the domain of Tactical Route. Ontology 1 conceptualizes a data model storing tactical route information. Ontology 2 conceptualizes the planning and execution considerations for Tactical Routes. Even though both ontologies are describing the same concept, they are conceptualizing Tactical Route in totally different ways. The Data Model Ontology is defining the terms of a Tactical Route by the function they perform in the data model, such as a primary key. The other ontology is describing the concepts as concrete factors used to describe a Tactical Route. The two ontologies, if merged or combined in total or linked by mapping a few terms or concepts would likely produce undesirable mismatch errors (See Figure 53).

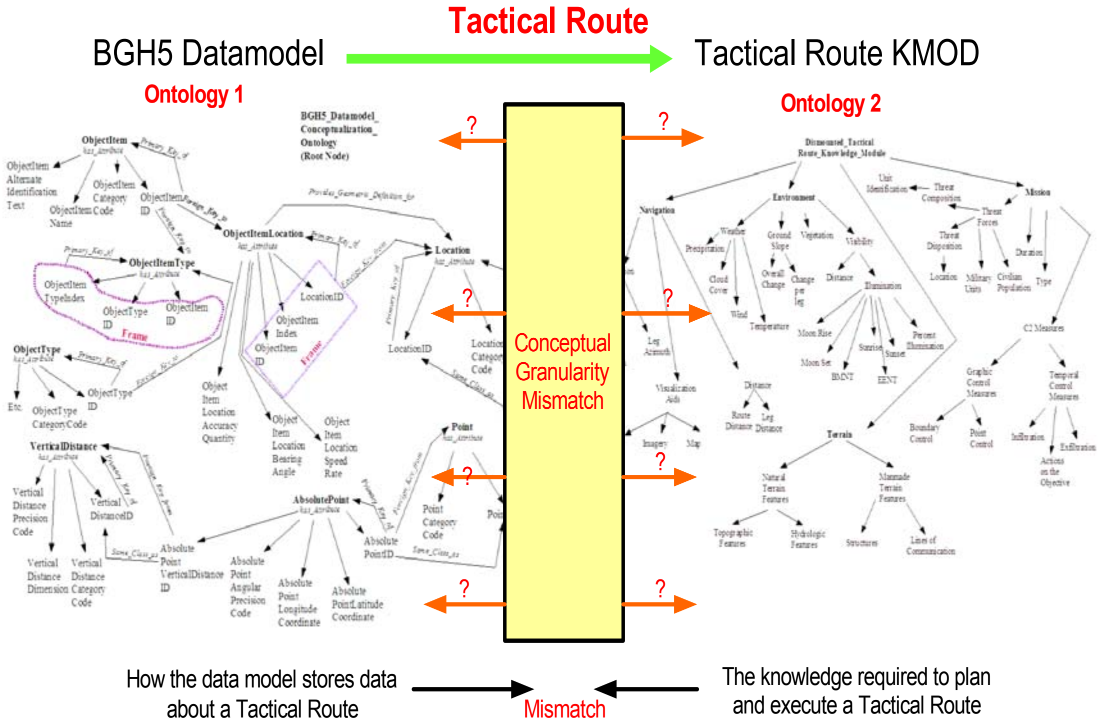


Figure 53. Conceptual Mismatch Example.

(3) Ontology Articulation Design Pattern. The Articulation Design Pattern, Figure 54, emerged as a result of the many problems associated with combining and merging ontologies. The articulation ontology advocates creating a new ontology from two or more existing ontologies and linking to the desired concepts through the articulation surrogate (Klein, 3). The articulation essentially takes the logical intersection between two existing ontologies and places the results in a third ontology or the articulation. This allows the entities in the articulation to retain their original implications and have them extended by the articulation. The articulation eliminates through separation of concerns any mismatch that may occur by keeping the two original ontologies apart. It is worth noting that the articulation can also function as a simple mapping between ontologies providing no mismatches exist between the entities.

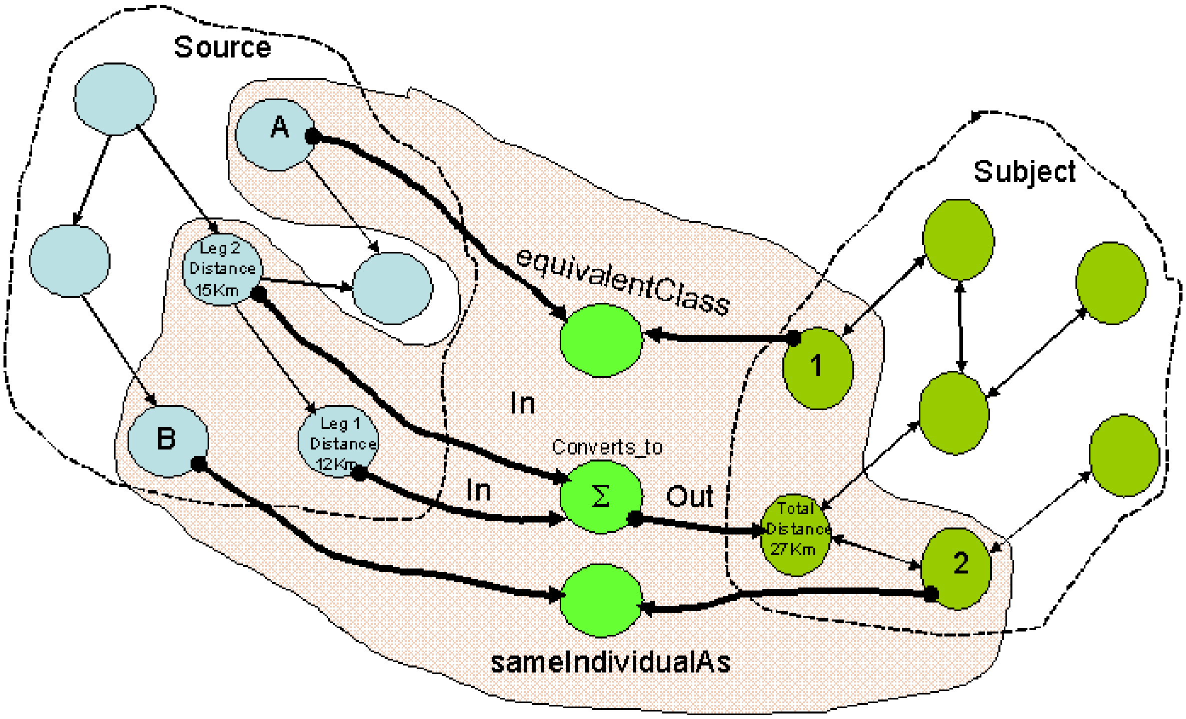
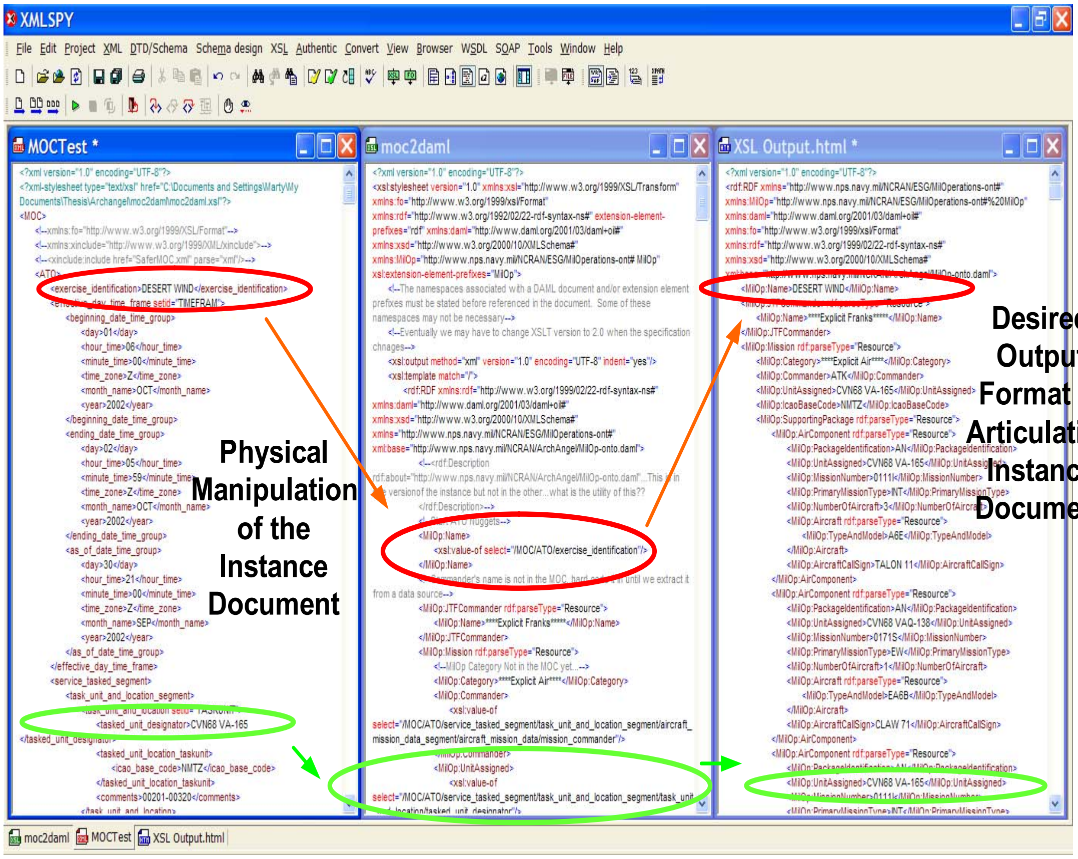


Figure 54. Articulation Abstraction.

A critical mechanism enhancing the function of the articulation ontology by providing the instance level transforms is the XSLT. While the Articulation Ontology syntactically, logically and computationally makes assertions about the articulated concepts, a XSLT derived from the Articulation Ontology can physically manipulate literals, labels and other data transforms to further facilitate the articulation. Articulation mapping done 1 to 1 at the instance level suffers a very low reuse factor and are extremely tedious to construct if the ontology is large. Figure 55 illustrates the tedious nature of 1 to 1 mapping at the instance level.


XMLSPY
File Edit Project XML DTD/Schema Schema design XSL Authentic Convert View Browser WSDL SOAP Tools Window Help
MOCTest *
moc2daml
XSL Output.html *
Physical Manipulation of the Instance Document
Desired Output Format of Articulation Instance Document
moc2daml
MOCTest
XSL Output.html

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="C:\Documents and Settings\Marty\My
Documents\Thesis\Archangel\moc2daml\moc2daml.xsl"?>
<MOC>
    <!--xmlns:fo="http://www.w3.org/1999/XSL/Format"-->
    <!--xmlns:xinclude="http://www.w3.org/1999/XML/xinclude">-->
    <!--<xinclude:include href="SaferMOC.xml" parse="xml"/>-->
    <ATO>
        <exercise_identification>DESERT WIND</exercise_identification>
        <effective_day_time_frame setid="TIMEFRAM">
            <beginning_date_time_group>
                <day>01</day>
                <hour_time>06</hour_time>
                <minute_time>00</minute_time>
                <time_zone>Z</time_zone>
                <month_name>OCT</month_name>
                <year>2002</year>
            </beginning_date_time_group>
            <ending_date_time_group>
                <day>02</day>
                <hour_time>05</hour_time>
                <minute_time>59</minute_time>
                <time_zone>Z</time_zone>
                <month_name>OCT</month_name>
                <year>2002</year>
            </ending_date_time_group>
            <as_of_date_time_group>
                <day>30</day>
                <hour_time>21</hour_time>
                <minute_time>00</minute_time>
                <time_zone>Z</time_zone>
                <month_name>SEP</month_name>
                <year>2002</year>
            </as_of_date_time_group>
        </effective_day_time_frame>
        <service_tasked_segment>
            <task_unit_and_location_segment>
                <task_unit_and_location setid="TASKUNIT">
                    <tasked_unit_designator>CVN68 VA-165
</tasked_unit_designator>
                    <tasked_unit_location_taskunit>
                        <icao_base_code>NMTZ</icao_base_code>
                    </tasked_unit_location_taskunit>
                    <comments>00201-00320</comments>
                </task_unit_and_location>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/xsl/Format"
xmlns:rdf="http://www.w3.org/1992/02/22-rdf-syntax-ns#" extension-element-
prefixes="rdf" xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
xmlns:MilOp="http://www.nps.navy.mil/NCRAN/ESG/MilOperations-ont# MilOp"
xsl:extension-element-prefixes="MilOp">
    <!--The namespaces associated with a DAML document and/or extension element
prefixes must be stated before referenced in the document.  Some of these
namespaces may not be necessary-->
    <!--Eventually we may have to change XSLT version to 2.0 when the specification
chnages-->
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/">
        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
xmlns="http://www.nps.navy.mil/NCRAN/ESG/MilOperations-ont#"
xml:base="http://www.nps.navy.mil/NCRAN/ArchAngel/MilOp-onto.daml">
            <!--<rdf:Description
rdf:about="http://www.nps.navy.mil/NCRAN/ArchAngel/MilOp-onto.daml"...This is in
one versionof the instance but not in the other...what is the utility of this??
            </rdf:Description>-->
            <!--Start ATO Nuggets-->
            <MilOp:Name>
                <xsl:value-of select="/MOC/ATO/exercise_identification"/>
            </MilOp:Name>
            <!--Commander's name is not in the MOC..hard code it in until we extract it
from a data source-->
            <MilOp:JTFCommander rdf:parseType="Resource">
                <MilOp:Name>****Explicit Franks*****</MilOp:Name>
            </MilOp:JTFCommander>
            <MilOp:Mission rdf:parseType="Resource">
                <!--MilOp Category Not in the MOC yet...-->
                <MilOp:Category>****Explicit Air****</MilOp:Category>
                <MilOp:Commander>
                    <xsl:value-of
select="/MOC/ATO/service_tasked_segment/task_unit_and_location_segment/aircraft_
mission_data_segment/aircraft_mission_data/mission_commander"/>
                </MilOp:Commander>
                <MilOp:UnitAssigned>
                    <xsl:value-of
select="/MOC/ATO/service_tasked_segment/task_unit_and_location_segment/task_unit
_and_location/tasked_unit_designator"/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns="http://www.nps.navy.mil/NCRAN/ESG/MilOperations-ont#"
xmlns:MilOp="http://www.nps.navy.mil/NCRAN/ESG/MilOperations-ont#%20MilOp"
xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:fo="http://www.w3.org/1999/xsl/Format"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema#"
xml:base="http://www.nps.navy.mil/NCRAN/ArchAngel/MilOp-onto.daml">
    <MilOp:Name>DESERT WIND</MilOp:Name>
    <MilOp:JTFCommander rdf:parseType="Resource">
        <MilOp:Name>****Explicit Franks*****</MilOp:Name>
    </MilOp:JTFCommander>
    <MilOp:Mission rdf:parseType="Resource">
        <MilOp:Category>****Explicit Air****</MilOp:Category>
        <MilOp:Commander>ATK</MilOp:Commander>
        <MilOp:UnitAssigned>CVN68 VA-165</MilOp:UnitAssigned>
        <MilOp:IcaoBaseCode>NMTZ</MilOp:IcaoBaseCode>
        <MilOp:SupportingPackage rdf:parseType="Resource">
            <MilOp:AirComponent rdf:parseType="Resource">
                <MilOp:PackageIdentification>AN</MilOp:PackageIdentification>
                <MilOp:UnitAssigned>CVN68 VA-165</MilOp:UnitAssigned>
                <MilOp:MissionNumber>0111I</MilOp:MissionNumber>
                <MilOp:PrimaryMissionType>INT</MilOp:PrimaryMissionType>
                <MilOp:NumberOfAircraft>3</MilOp:NumberOfAircraft>
                <MilOp:Aircraft rdf:parseType="Resource">
                    <MilOp:TypeAndModel>A6E</MilOp:TypeAndModel>
                </MilOp:Aircraft>
                <MilOp:AircraftCallSign>TALON 11</MilOp:AircraftCallSign>
            </MilOp:AirComponent>
            <MilOp:AirComponent rdf:parseType="Resource">
                <MilOp:PackageIdentification>AN</MilOp:PackageIdentification>
                <MilOp:UnitAssigned>CVN68 VAQ-138</MilOp:UnitAssigned>
                <MilOp:MissionNumber>0171S</MilOp:MissionNumber>
                <MilOp:PrimaryMissionType>EW</MilOp:PrimaryMissionType>
                <MilOp:NumberOfAircraft>1</MilOp:NumberOfAircraft>
                <MilOp:Aircraft rdf:parseType="Resource">
                    <MilOp:TypeAndModel>EA6B</MilOp:TypeAndModel>
                </MilOp:Aircraft>
                <MilOp:AircraftCallSign>CLAW 71</MilOp:AircraftCallSign>
            </MilOp:AirComponent>
            <MilOp:AirComponent rdf:parseType="Resource">
                <MilOp:PackageIdentification>AN</MilOp:PackageIdentification>
                <MilOp:UnitAssigned>CVN68 VA-165</MilOp:UnitAssigned>
                <MilOp:MissionNumber>0111I</MilOp:MissionNumber>
                <MilOp:PrimaryMissionType>INT</MilOp:PrimaryMissionType>
```

Figure 55. XSLT Articulation Instance Transform.

b. Knowledge Module

The second ontology design pattern that emerged from this research was the idea of Domain Oriented Knowledge Structuring (DOKS) within an ontology. DOKS will be discussed at length in the Knowledge Base Chapter as a technique to modularize knowledge into a series of autonomous knowledge bases organized similarly to how a human would organize knowledge (Wachsmuth, 1991, 1). From an ontology design perspective the DOKS principles are used to group relevant knowledge required for a specific purpose and capture it in an ontology for use in a specialized domain. As a result, the knowledge is structured in a way oriented at the very specific problem space of a domain (Wachsmuth, 1991, 5). Knowledge Modules should be used to assist with organizing large or highly specialized ontologies into logical categories to help speed access and search time for the user.

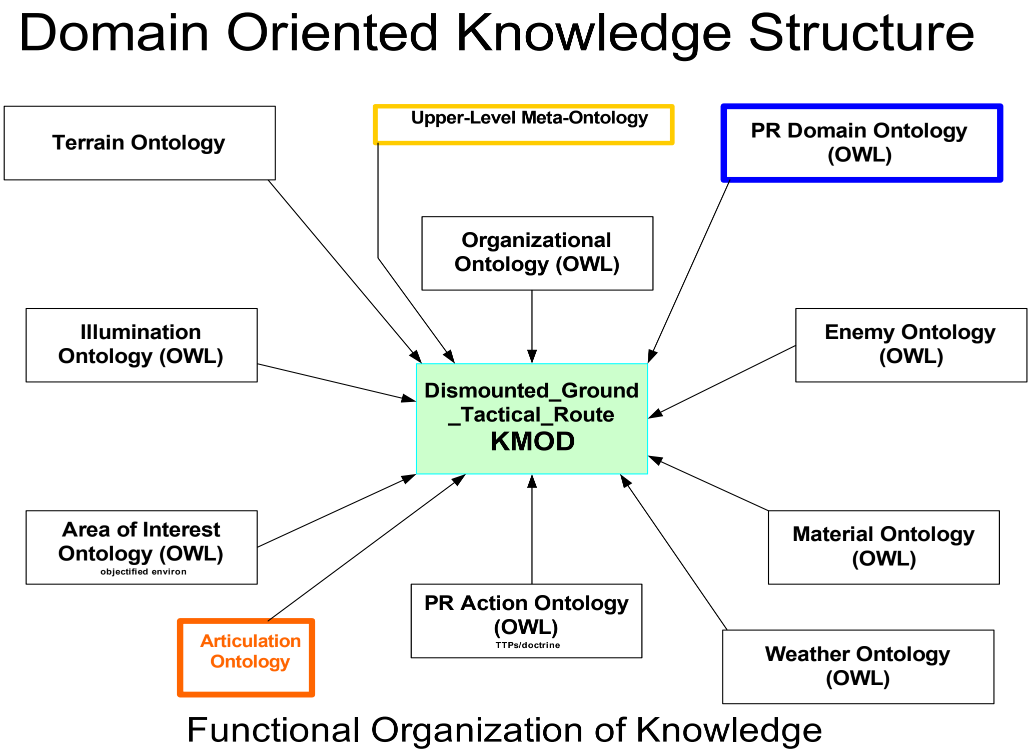
The DOKS concept originated from an effort to design a knowledge base to recognize and understand natural language text from tourists using the Dusseldorf Tour Guide (Wachsmuth, 1991, 1). The background knowledge required to recognize all possible usages of text applicable to this undertaking was enormous and performance was critical. The design team understood that large knowledge bases were unavoidable for the complexity of the problem and took to proposing modules of knowledge called knowledge packets for specifying when a piece of knowledge should be accessed (Wachsmuth, 1991, 3). As a result, Knowledge Modules (KMOD) were introduced as a mechanism to effectively group knowledge statements that belong together into autonomous packages.

With this idea in mind the similarities can be drawn from their large complex problem to any problem requiring the implementation of ontologies. Instead of designing an ontology to make runtime imports from many different ontologies, establish multi-disciplined knowledge module ontologies containing prepackaged information tailored to specific problem sets. This idea is particularly applicable and well suited to problem spaces in the military domain.

In the future, the goal is to dynamically create knowledge modules on demand with the most current knowledge to the specification of the ontology. In the interim the knowledge modules are formed and stored in a static state until they are called to use. The knowledge module consists of an ontology and a set of instances tailored to a specific problem. Let us look at a concrete example in the Military Domain regarding a KMOD Ontology for Dismounted Ground Tactical Routes (DGTR) (See Figure 56).

The problem space of DGTR is highly specialized. The knowledge required to plan and execute a DGTR is very different from the knowledge required to plan and execute a Mounted Ground Tactical Route or an Air Route for reference. Because of the specialization required it made sense to use the DOKS knowledge module concepts to organize the knowledge required to plan and execute a DGTR. As a result, when a DGTR is being planned the ontology and the instances of the DGTR KMOD would be recalled to support the application managing the DGTR planning. The DGTR KMOD would contain all knowledge applicable to planning and executing a DGTR.

In order to formulate a DGTR KMOD, an intensive knowledge acquisition effort was undertaken to elicit how experts in this area planned, executed and perceived DGTR (Stine, 2000, 15). We were able to use research done for a previous thesis to supply the required background knowledge from what should be included in the KMOD to how the KMOD was organized. Stine's research supplied us with an informal Restricted Vocabulary captured directly from the interviews with the domain experts. This Restricted Vocabulary was incorporated into the ontology design. Leveraging Stine's research we were able to conceptualize the DGTR KMOD. Let us look at the supporting components of the DGTR KMOD to see what the likely content of the KMOD could be.



Domain Oriented Knowledge Structure

Functional Organization of Knowledge

Figure 56. DGTR KMOD Components Example.

Now that we see the structure of the supporting knowledge required by the DGTR KMOD, lets examine the KMOD Domain Concept Tree the ontology was derived from (See Figure 57). When looking at the terms notice the usage of the Restricted Vocabulary and how the taxonomy is structured to import supporting knowledge of the categories shown above as it is developed. The red dots indicate a position in the taxonomy where a supporting ontology will be inserted. As one might expect KMODS are much more difficult to design and requires large amounts of domain expertise decide on the concepts to be included in the ontology

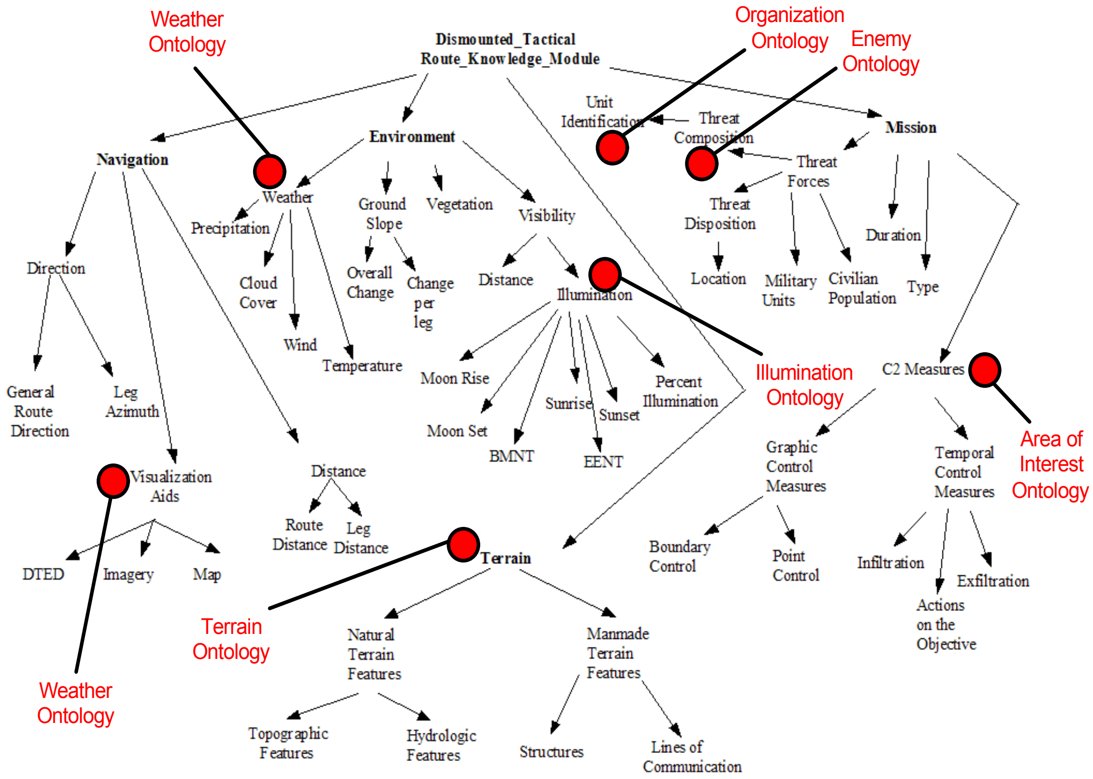


Figure 57. DGTR KMOD Supporting Components.

E. SUMMARY

Ontologies are an artifact composed of classes, properties and relations forming an aspect of the SWEB that will allow machines to interpret content and reason about it. The task of designing ontologies can be traced back to Plato and Aristotle as they attempted to classify objects of the world. An ontology is a specification of a conceptualization where a conceptualization is some abstract, simplified perception of the world we wish to represent for some purpose (Gruber, 1993, 1-2). Ontologies must be designed for a specific purpose and applied to a specific problem space for the best chances of implementation and design success. Ontologies can be formally written in many different knowledge representation languages, but OWL will likely be adopted by the W3C as its recommendation and become the de facto standard. OWL is a description logic based language that can be further subdivided into three specialized languages whose usage depended on the requirements of the user. In order for an ontology to be implemented successfully it takes commitment by the domain members to the lexical terms, definitions and concepts. Without commitment the likelihood of another application reusing or extending the ontology is very low. There are several design methodologies a developer can use when designing ontologies, but only one methodology explicitly advocates feasibility study and requirements generation. The tool support for ontologies is rudimentary at the present time and lacks the necessary validation and verification mechanisms to make the process of merging, combining and articulating ontologies user friendly. The realization of the SWEB will depend on the adoption, design, implementation and reuse of ontologies across the Web. Once machines can interpret web content the industries and agencies adopting the SWEB will realize the first portions of Return on Investment (ROI).

THIS PAGE INTENTIONALLY LEFT BLANK

VII. SWEB KNOWLEDGE BASE

A. BACKGROUND

The purpose of this chapter is to demonstrate the importance of the network to the Knowledge Base of a SWEB application. We will review design patterns and considerations contrasting the Knowledge Base (KB) of an SWEB application with the KBs supporting the Expert Systems (ES) of the 1990's. We will demonstrate how the SWEB KB can employ mechanisms to reason and apply rules against enabled content and discuss techniques for design and organization.

The term Knowledge Base (KB) originated from the early efforts of Artificial Intelligence to replicate human decision making (Kurzwiel, 1990, 292). The KB was intended to capture the ideas, concepts, descriptions, constraints, uncertainties and relationships of the domain. The KB was to replicate the knowledge a domain expert would use during the course of solving a domain related problem (Kurzweil, 1990, 292) (Marakas, 1999, 242). The design pattern that emerged for a KB was one of a centralized repository of codified, machine readable knowledge created detail by detail, relationship by relationship, by highly skilled human, knowledge engineers (Dean, 2003, 17) (Kurzweil, 1999, 292). To be effective, these KBs were supported by vast databases storing every conceivable fact the human designers could foresee being required to support the decision making process for a specific domain. The KB was the brains of what we know today as Expert Systems (ES).

One of the first ES, DENDRAL, began development in early 1965 and continued on through the mid-1970s. The success of DENDRAL and its design methodologies gave rise to the popularity of the ES industry during the 1980s (Kurzweil, 1999, 294). ES technology was developed for applications in the fields of medicine, insurance, energy, and the automotive industries to name a few (Kurzweil, 1999, 300). While certain industries experienced success in specific domains, the time to construct a KB, the expertise required and the physical storage capacity necessary began to reveal severe limitations in the scope of ES applicability.

Today, as the SWEB and its technologies seek mainstream adoption, the concept of a KB is finding new life and is being redefined. The KB of SWEB applications will be expected to function in a similar capacity as its predecessor in the ES field and will undoubtedly suffer many of the same problems. The SWEB, however, will offer the new generation of the KB the unique ability to leverage the knowledge contained in the vast networks of the World Wide Web. Let us be reminded that the goal of the SWEB is to link ontologies, make data and knowledge stores enabled by common meaning and platform independence so that it is available on demand. How will the "networked" KB of the SWEB be designed? Will the KB still be required to store all potential facts to support domain decisions a priori? The networks comprising the World Wide Web and the technologies enabling the capability to achieve common meaning are truly creating some interesting options.

This chapter is intended to guide the reader in attaining a better understanding of the functions and interactions of a traditional KB, as well as a "networked" KB, as it will likely occur in SWEB applications. We will discuss the KB in the traditional AI terms from its definition, design criteria, components, organization. This discussion will rely on the recollection of many of the foundational concepts we have discussed in early chapters as the KB is where all the concepts converge.

B. KNOWLEDGE BASE DEFINITION

The traditional definition of a KB varies between the expert and the context for which it was formed. Maedche et al define a KB as a collection of object descriptions that refer to an ontology (Maedche et al, 2003, 325). We will refer to this aspect of their definition as the concise portion. The definition is then extended by exhaustive enumeration, just as an OWL class axiom might define a class by extension[94]. It is worth examining this enumeration in detail and comparing it to Noy and McGuinness's definition of KB to identify the points of difference.

Noy and McGuinness define KB as an ontology along with its set of instances and further state that a fine line exists where an ontology ends and the KB begins (Noy et al,

[94] See Chapter VI.

2001, 3). That said; let us continue with Maedche et al's definition to see if we can determine if it is this fine line they were hoping to expose by the enumeration included in their definition. Maedche et al's enumeration of what a KB contains follows:

- Lexicon-set of signs for instances (ontology)
- Reference function-linking instances to classes and properties they correspond to (ontology)
- Set of instances
- Membership function relating instances to concepts (ontology and instances)
- Instantiated relations (ontology and instances)
- Axioms (some in the ontology)
- Reference to an ontology (Instances)

In the concise portion of Maedche et al's definition we find agreement among both experts' definitions that an ontology is a part of the knowledge base. Curiously, the distinction Noy and McGuinness make treats the ontology as the mechanism in which the domain is exclusively modeled using a knowledge representation (KR) language, such as OWL, that embeds the relations and references within the representation. We have annotated the enumeration above in terms of Noy and McGuinness's definition. In parentheses we have labeled whether the instances or ontology handles the function. Maedche at al leave this open ended and generic. If we take Noy and McGuinness's definition and assume the ontology handles the task of modeling the domain, then we have effectively consolidated bullets 1-2, 4, 6-7 of the KB enumeration into the responsibility of the ontology, leaving only the bullets related to instances as stand alone concepts[95].

Obviously, Maedche et al were revealing a design choice as to where to model the domain, the KB or the ontology (Maedche *et al.,* 2003, 325). This choice brings us to our first important design decision. If the line between the ontology and the KB is so fine, according to Noy and McGuinness, does it really matter where the domain is modeled? We would submit that in a KB of the traditional sense found in an ES in the 1980s it may

[95] An argument can be made for bullet 6, axioms, to be considered a standalone concept as well. This categorization would be dependent on whether or not the application required a more specific rule set outside of what the ontology could provide.

have mattered, but today with the prevalence of networks the choice to model the domain in the KB vice the ontology is not a significant performance distinction as long as the KB is exposed and accessible to the network. One caution, however, might be found in the extensibility aspect of the domain model. If the KB is used to model portions of the domain instead of exposing it through an ontology, the KB must also be exposed to the extent it was used to model the domain if the domain theory is to remain extensible. Therefore, the extensibility of the domain theory and the KR can be diminished and made less transferable if care is not taken ensure the domain theory can still be exposed. With SWEB applications we do not see this as a huge distinction between the definitions; however, in these cases we recommend careful consideration be given so as not to subdue the extensibility of the domain theory and prevent it from being reused and extended. As a counterpoint, there are likely applications where the KB must be used to model portions of the domain. Let us continue and make two more points about the definition of KB.

The two previous definitions, though seemingly complete, did not make mention of belief, reasoning or truth. These three aspects of a KB are essential to understanding how a KB functions and a case can be made for inclusion of these aspects into any definition. Levesque suggests a KB is a collection of symbolic structures representing what it believes and reasons with during operations on the system (Levesques, 2000, 8). The symbolic structures can be equated and mapped with straightforward precision to the ontology and its instances. To help us understand the "belief and reason" portion, let us look at one last definition to gain clarification.

Wachsputh's definition states a KB is a set of identifiable statements, each of which interrelates domain specific concepts and asserts something held for true in a modeled world (Wachsputh, 1991, 5). The latter definition substitutes truth where Levesques had belief. As we shall see in our upcoming discussion, what a KB knows is what is true[96]. That which is true to a KB is considered its belief. What it knows and believes is therefore what it reasons with during the course of operation. To further support our discussion about where the ontology ends and the KB begins, we know we can assert what we believe to be true in a domain either in an ontology or the KB. As

[96] This is not to be confused with the truth of the fact with respect to the world however. The KB can contain false facts it just does not know they are false (Truth Axiom covered in the reasoning chapter).

such, as long as the belief of a domain is clearly and consistently asserted somewhere, whether the KB or the ontology, and proper care is taken to ensure extensibility, then the system will function.

Finally, let us analyze the critical aspects of a KB we have exposed from our analysis of the four definitions. According to our analysis a KB should include:

- Ontology
- Instances of the ontology
- A set of facts believed to be true to reason over
- Network interface

When taken from the traditional perspective one might stop our analysis and settle with the ontology, instances of the ontology and a set of facts to reason over. But the network is the critical component of the KB of the SWEB. As such, we added it to our enumeration of required KB components (See Figure 58).

As we made reference to above, SWEB applications will rely on the networks of the World Wide Web to provide data, information, and knowledge on demand. The value of a KB in a SWEB application is not necessarily the contents of the KB at present, but the speed and the knowledge sources it can discover, access, enable and reason against available from the network. These properties are dependent on reliable networks and deep interoperability provided by common meaning established by interconnected ontologies. The enabling technologies and mechanisms discussed in the Data Sources and Distributed Computing chapters are intended bring about reliability in the network and data-level interoperability. While the centralized knowledge base is still very much the comfort level of designers and is used in most knowledge based systems, the KBs supporting SWEB applications should include the network as an essential component of its KB (Dean, 2002, 17). After all, the knowledge is in the network.

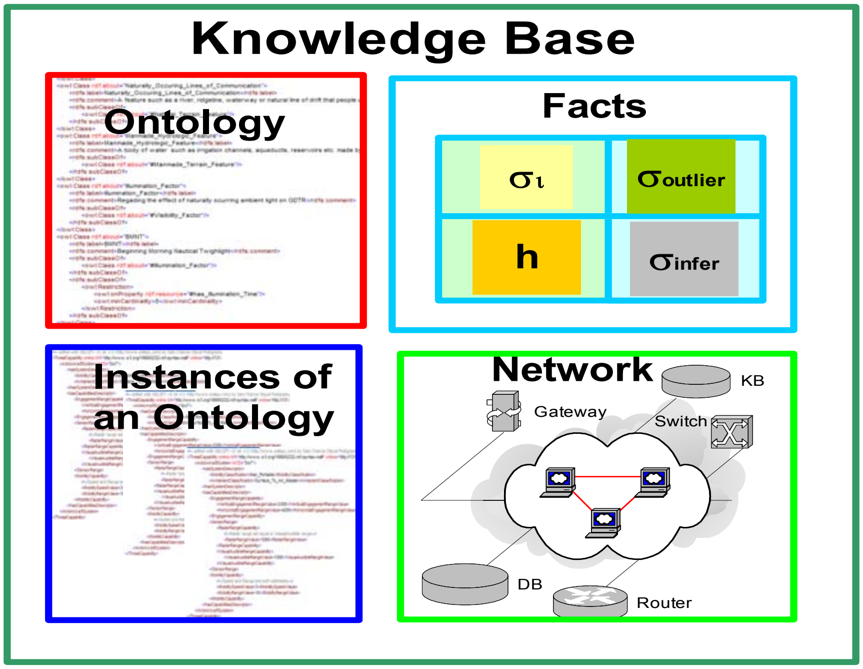


Figure 58. SWEB Knowledge Base Schematic.

C. KNOWLEDGE BASE COMPONENTS

As we might have discerned from the definitions, a given KB contains certain entities and concepts consistent across all KBs. We will discuss each of these entities in detail in the section that follows. These entities can be further classified into "*what a KB has*" and "*what a KB does*". The following assertions pertain to a generic KB and will function to bound our upcoming discussion:

- All KBs have facts, propositions, belief, and representation.
- All KBs must reason, must support human or programmatic interaction and be maintained.

Let us start our discussion with what a KB has.

1. Facts

The most basic element of a KB is a fact. A fact is simply a statement about the domain of discourse the KB takes as true at the time of assertion (Fagin, 1995, 116) (Levesque, 2000, 3). Facts can be either asserted in the form of a TELL statement to the KB or retrieved from the KB in the form of an ASK statement. Facts are formed into sentences and stored in the KB as such. A KB sentence must take the form of a subject, predicate and object. Sentences are the subject of reasoning operations within a KB.

Sentences communicate ideas or abstract entities that can be true or false, right or wrong (Levesque, 2000, 3). These abstract entities are called propositions. Sentences along with their propositions can be told to a KB and they will either be supported as logically consistent, in which case they are accepted into the KB, or found logically inconsistent in which case they return errors when reasoned against.

2. Belief

The KB's belief is the manner in which the KB implicitly imagines the multiple ways the world can be true (Levesque, 2000, 4). To continue we must review the two views of knowledge from the Knowledge Chapter, the "one world" and the "possible worlds" theories. If the KB aligns with the one world theory we can think of the knowledge contained within the KB as a set of propositions we hold to be true against the model of that one world. This perspective of the KB's propositions is consistent with our one world or extensional knowledge view. To review, extensional knowledge is largely assertional knowledge about individuals and tends to be somewhat more dynamic in nature when compared to its counterpart intensional knowledge (Gardenfors, 2000, 152).

The alternative belief of a KB can be thought of in terms of the possible ways in which the world can be. Each of these different ways is considered a truth condition and takes the form of intensional knowledge. Intensional knowledge is largely declarations describing the general properties of concepts and tends to be more timeless. Intensionally aligned propositions will be classified by the KB into groups of propositions that are incorrect, those that are consistent with the way the world is according to the logic of the KB, and finally the way the world really is. As we also recall, the KB only knows the propositions it believes to be true and has no way to determine the true state, or the way the world really is. Regardless of the proposition's alignment, extensional or intensional, the KB's belief is the way in which the KB's logic views the world. There are several knowledge axioms guiding the belief and truths of a KB that we will discuss at length in the reasoning section.

3. Representation

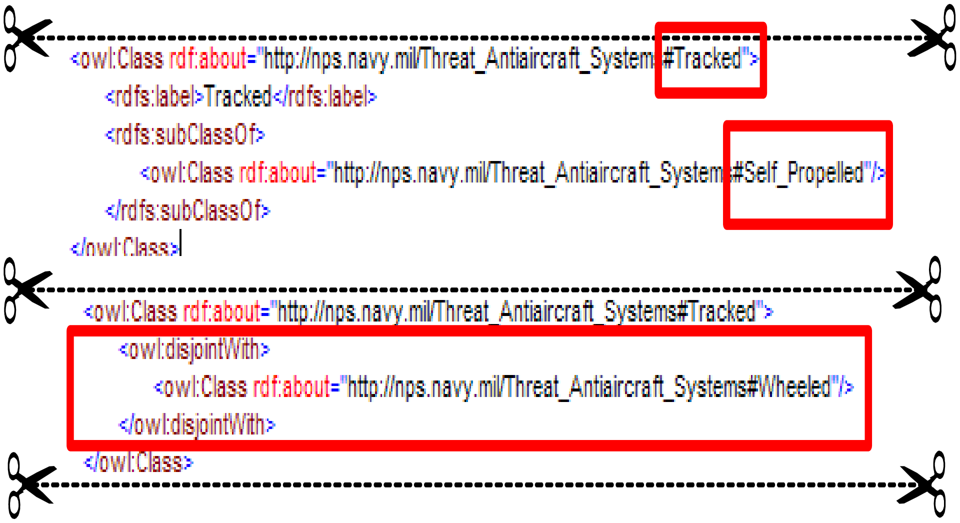
In the previous sections we established that a KB has a set of facts, formed into sentences that communicate ideas called propositions. The propositions created by combining the facts of a KB are the concepts existing in the domain of discourse. The KB is told these facts about the external world, either programmatically or via human input. The KB can then be asked queries against those facts or propositions (Fagin, 116, 1995). Since we have established that the ontology is part of the KB and we know that the ontology must be represented by a Knowledge Representation (KR) language, it is not difficult to conclude that the content within the KB, from the facts to the propositions, must also be represented and compatible with the ontology. KR, as described by Levesque, is concerned with using formal symbols to represent a collection of propositions believed by some KB[97] (Levesque, 2000, 6). Earlier in this work we discussed OWL as the soon to be W3C's recommendation to represent ontologies. This recommendation also extends to the instances or facts of the KB and will be the terms in which we will describe KR in this work.

When we examine the example OWL representation of the below propositions in Figure 59, we notice the term "Tracked[98]" in our proposition can easily be described further in an infinite number of supporting ontologies with varying degrees of specificity. In our KB, however; we have described "Tracked" consistent with the purposes for which our users require it described. Our description is simply not wheeled and self propelled. We know there are an infinite number of propositions that can be believed about the concept "Tracked," of which we are only representing two (Levesque, 2000, 6):

- not wheeled
- self propelled

[97] Levesque uses the term agent. We substituted KB for agent as that is the topic of our conversation.

[98] When we refer to "Tracked" we are referring to the tread of a tank or armored personnel carrier.



```
<owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Tracked">
    <rdfs:label>Tracked</rdfs:label>
    <rdfs:subClassOf>
        <owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Self_Propelled"/>
    </rdfs:subClassOf>
</owl:Class>
```

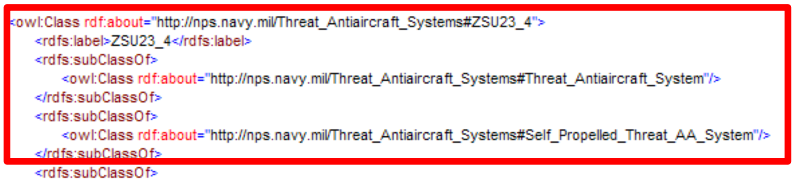
```
<owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Tracked">
    <owl:disjointWith>
        <owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Wheeled"/>
    </owl:disjointWith>
</owl:Class>
```

Figure 59. Tracked Class Representation Example.

According to Levesque, it is at this point where "reasoning must bridge the gap between what is represented and the full set of propositions believed by the KB" (Levesque, 2000, 6). It is important to note that the reasoning can only occur to the extent of the completeness of the represented concepts within a domain, as we shall see. We will demonstrate how reasoning can accomplish this in a later section.

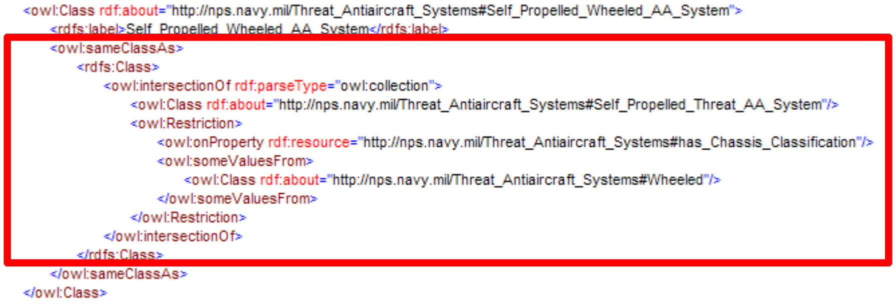
In Figure 60 we have listed some propositions containing facts asserted to our KB both in the form of OWL marked up concepts and English sentences. Both communicate the same ideas or concepts and are formed by the same set of facts. The OWL concepts differ from the English sentence by enabling the computer to interpret the meaning of the concept relative to relationships and comparisons with other marked up concepts within the domain. The computer can accomplish this through interpreting the description logic underpinnings and the formal semantics abstracted by the OWL markup. The accuracy of the interpretation of the concepts by the computer is directly influenced by the completeness and accuracy of the KR representing the concepts (Marakas, 1999, 264). This is due to the fact that the KR is just a representation of the knowledge; it is not the knowledge itself (Marakas, 1999, 264). KR is a surrogate for entities that cannot be represented in a computer such that it encodes knowledge (Sowa, 2000, 135). The KR, therefore, is the computable model that allows the ontology and the logic to be implemented within applications (Sowa, 2000, xii). Currently, with the exception of very few specialized systems, the English language cannot be interpreted by computers[99], one of the reasons KR is required.

English Sentence/Proposition (S1)
ZSU 23-4 anti-aircraft systems are Self-Propelled (SP)



```
<owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#ZSU23_4">
    <rdfs:label>ZSU23_4</rdfs:label>
    <rdfs:subClassOf>
        <owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Threat_Antiaircraft_System"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Self_Propelled_Threat_AA_System"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
```

English Sentence/Proposition (S2)
Wheeled anti-aircraft systems are Self-Propelled (SP)



```
<owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Self_Propelled_Wheeled_AA_System">
    <rdfs:label>Self_Propelled_Wheeled_AA_System</rdfs:label>
    <owl:sameClassAs>
        <rdfs:Class>
            <owl:intersectionOf rdf:parseType="owl:collection">
                <owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Self_Propelled_Threat_AA_System"/>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="http://nps.navy.mil/Threat_Antiaircraft_Systems#has_Chassis_Classification"/>
                    <owl:someValuesFrom>
                        <owl:Class rdf:about="http://nps.navy.mil/Threat_Antiaircraft_Systems#Wheeled"/>
                    </owl:someValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
        </rdfs:Class>
    </owl:sameClassAs>
</owl:Class>
```

Figure 60. OWL Knowledge Representation Example.

4. Reasoning

Reasoning is the action coming from what a KB believes true about the world that is not explicitly stated (Levesque, 2000, 10). As we said above reasoning is the bridge between what we have explicitly represented in a KB and what can be included in the set of believable propositions as determined by the logic and representation of the KB. If the KB was unable to reason, it would function much like a database only able to return explicitly stated and stored facts and propositions. The KR and the logic embedded within establish the belief of the KB through the embedded logic of the KR and enable a KB to reason about its beliefs (Levesque, 2000, 10). As we briefly mentioned above, the

[99] Natural Language Processing (NLP) is another important emerging field.

KB bases its belief on what it is told (facts) and what it takes to be true. How do we know what is true? To establish this concept let us examine the Knowledge or Truth Axiom.

a. Truth Axiom

The Truth Axiom, as described by Fagin, is one of the distinguishing characteristics between knowledge and belief (Fagin, 1995, 32). The Truth Axiom states that a KB or system, bases its belief on what it has represented. We know from our discussion above that the KR at the atomic level represents facts. If facts are represented, then facts are the basis for the KB's belief as we established above. A fact, by definition, must be believed to be true at the time of assertion to the KB (Fagin, 1995, 33). Although, as Fagin states, a KB may not know facts that are true, it is the case that if a KB knows a fact that it must be true. In the case of our Anti-aircraft ontology two facts that we know our KB knows are listed in Table 7 below.

Facts	**Condition**
(F1) Tracked anti-aircraft systems are Self-Propelled (F2) Any Self-Propelled anti-aircraft system can also be wheeled	True

Table 7. Example Facts.

The Truth Axiom distinguishes between knowledge and belief by further stating, "A KB may have false beliefs, but it cannot know that something is false (Fagin, 1995, 32). This may be confusing, but if we decompose the statement we can understand why the statement must be true. To make this point let us restate what we asserted above in slightly different terms. We stated that a KB cannot know something that is not true. A restated version in different terms follows that a KB cannot know something that is false. This follows because "not true" is a synonym for "false". To continue, a KB only knows facts, and by definition facts are necessarily true. So, a KB can falsely believe something, but it cannot know what it falsely believes is false, since it only knows facts, and facts must be true by definition. Establishing that facts are true and that the belief of the KB is based on its facts is a central concept to understanding reasoning.

b. Entailment

The product of reasoning itself is referred to as an entailment. An entailment is defined by Merriam-Webster's Dictionary as "Something that is inferred or deduced." Levesque describes an entailment by propositions represented by a set of sentences (S) that entail the proposition represented by sentence (P) when the truth of (P) is implicit in the truth of (S) (Levesque, 2000, 10). To continue our example from above let us apply this to assist us in understanding the concept of entailment.

We asserted the propositions contained in Table 8. We know that since the propositions consist of facts, these propositions must be believed to be true. As such, let us form a logical entailment from the propositions. A likely entailment would be Self-Propelled systems are wheeled and tracked. We can easily see that the truth of our entailment is in fact implicit in the truth of our set of sentences (S). Therefore, the KB should believe the entailment because it follows from two propositions that are explicitly represented (Levesque, 2000, 13). To simplify, the reasoning mechanism in a KB must compute all possible entailments.

	Sentence	**Condition**
Sentence Set (S)	(S1)Tracked anti-aircraft systems are Self-Propelled (S2) Any Self-Propelled anti-aircraft system can also be wheeled	True
Sentence (P)	Self-Propelled systems can be wheeled or tracked[100] **(Entailment)**	True

Table 8. Entailment Example.

c. Computer Reasoning

Computers have the ability to reason in many different ways, some of which are complex and others are quite simple. For the purposes of this work our intent is to highlight a few of the basic ways computers reason in Description Logic (DL) based constructs and then show how reasoning can be applied to the ontology and the KB in a

[100] We also must make the assumption that Self-Propelled Systems cannot be both wheeled and tracked. This would likely be handled with a Disjoint Axiom. If we make this assumption our entailment is valid.

military example. This section will highlight many of the concepts and ideas we have discussed so far. Before we begin with the analysis of the example let us examine the basic principals of computer reasoning.

(1) Modus Ponens. Modus Ponens uses substitution and what it knows about the world from the facts in the KB to make very powerful inferences. Modus Ponens states that if A is true and A implies B (A→B) is true, then B is also true (Marakas, 1999, 242) (Russell, 1995, 269). Let us look at Table 9 to see how Modus Ponens applies to the Anti-aircraft example we started above. Table 9 illustrates the Domain Concept Tree from the ontology so we can establish traceability to where the truth conditions reside that Modus Ponens is referring to. For our examples all our rules and truth conditions are traceable to the description of the concepts in the ontology. In this exemplar we did not apply external rules. We will continue our discussion with the Threat Antiaircraft System example to illustrate the powerful nature of SWEB application reasoning against the class hierarchy of our ontology.

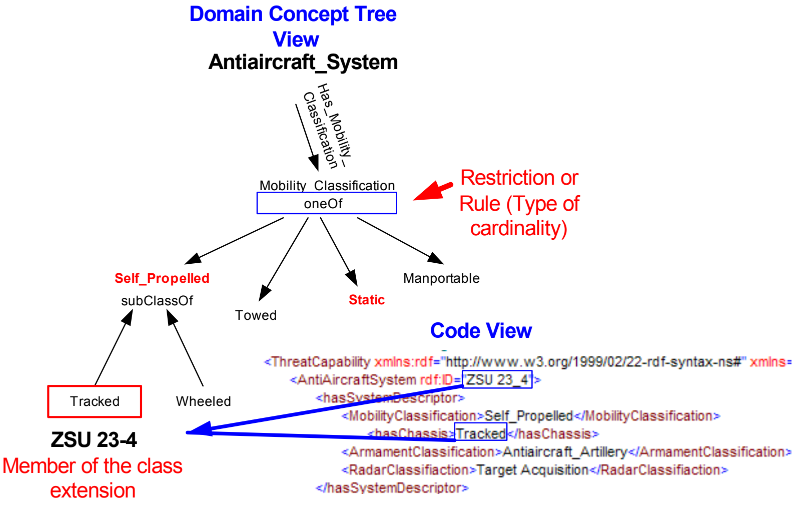
Name	**Sentence**	**Condition**
A	Tracked Anti-aircraft Systems are classified as Self-Propelled (SP) and not static	True
A→B	ZSU 23-4 *(known tracked system)* is classified as Self-Propelled (SP)	True
B	**ZSU 23-4 is not classified as Static**	**True**

Table 9. Modus Ponens.

By examining the domain concept tree and the snippet of markup from the ontology we can see how Modus Ponens can work inside the rules established by the ontology. If we refer to the proposition asserted by Sentence A above we see that Tracked Anti-aircraft systems are SP and not static. If we refer now to Figure 61 we can see this fact is true. In fact the restriction or rule imposed by this ontology states a Threat Anti-aircraft System can have only one Mobility Classification and therefore, if it is classified as Self-Propelled, it cannot also be classified as static. If we look further we can identify a subclass of the Self-Propelled Mobility Classification called Tracked. Therefore, the ontology established everything in Sentence A as true.

Let us now look at the representation to determine how A implies B (A→B). If we refer to the markup snippet from the instance document below, we observe the ZSU 23-4 system contains tag hasChassis with the value being Tracked. This asserts a ZSU 23-4 as a tracked system, and since it is a tracked system we know it is also Self-Propelled because tracked is a subclass of Self Propelled. Because it is Self-Propelled we know it cannot be static, because of the restriction allowing only one Movement Classification. Therefore, we know A implies B (A→B) is also true.

Finally, Modus Ponens allows us to state that the ZSU23-4 is not static even though it is not explicitly stated. This is a very simple example, but there is value gained from a SWEB application able to execute these types of simple operations on the fly. The classification example we just discussed is currently done by a human analyst. Now we understand where the truth conditions originate, let us look at the general procedures that apply our rules inside the inference engine.



```
<ThreatCapability xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns=
    <AntiAircraftSystem rdf:ID="ZSU 23_4">
        <hasSystemDescriptor>
            <MobilityClassification>Self_Propelled</MobilityClassification>
                <hasChassis>Tracked</hasChassis>
            <ArmamentClassification>Antiaircraft_Artillery</ArmamentClassification>
            <RadarClassifiaction>Target Acquisition</RadarClassifiaction>
        </hasSystemDescriptor>
```

Figure 61. Anti-aircraft Modus Ponens Example

(2) Chaining. Chaining is a basic procedure that applies the rules of deduction to produce a line of reasoning (Marakas, 1999, 242) (Russell, 1995, 272). There are two types of chaining, forward chaining and backwards chaining. It is important we can differentiate between these terms so we can also differentiate between the types of procedures employed by the various inference mechanisms.

Forward Chaining. Forward chaining is executed when a new fact or proposition is added to the KB. In forward chaining the inference begins with the assertion of a new fact and attempts to establish conclusions based on the new fact (Marakas, 1999, 242). The forward chaining procedure adds all sentences to the KB that can be inferred from the new fact (Russell, 1995, 274). Forward chaining slowly builds up a general picture of the problem space as new facts become available. Forward chaining is not directed at solving any particular problem and can be referred to as data-driven or data-directed (Russell, 1995, 274). To reinforce our understanding of the forward chaining procedure we will look at an example (See Table 10).

Table 10 illustrates the assertion of two new facts F1 and F2 to our KB. From F1 and F2 the Forward Chaining inference mechanism in our application immediately attempts to draw new conclusions from the new facts. From the new facts the inference mechanism was able to conclude C1. Again our forward chaining mechanism was not trying to solve a specific problem it was continually adding to the KB's picture from the fact assertions entered into the KB from the working memory or data input (Marinescu, 2002, 440).

Name	Sentence	Condition
New Facts Enter KB F1 and F2	(F1)Tracked anti-aircraft systems are Self-Propelled (F2) Any Self-Propelled anti-aircraft system can also be wheeled	True
New Conclusion (C1)	Self-Propelled anti-aircraft systems are wheeled and tracked	True

Table 10. Forward Chaining Example.

Backwards Chaining. Backwards chaining is designed to find all the answers to a question ASKed to a KB (Russell, 1995, 275). Backwards chaining is executed by checking to see if the ASK can be provided directly from the content of the KB (Russell, 1995, 274). Backwards chaining is referred to as goal directed because it starts with the goals and works backwards to find support for the goal (Marakas, 1999, 243). To help illustrate this point let us take our Modus Ponens example above and transform it into a backwards chaining problem (See Table 11).

A backwards chaining problem starts with the goal and in our example the goal is that the ZSU 23-4 is not classified as static. Next the backward chaining procedure would examine the facts and propositions within the KB to see if the goal can be supported by what the KB knows. In fact, in our KB we would be able to return the facts contained in Table 11 in support of our goal. Since these facts support our goal we can infer that our goal is true.

	Sentence	**Condition**
Goal	The ZSU 23-4 is not classified as Static	Unknown
Fact	**Sentences Supporting Goal**	
A	Anti-aircraft systems can have only one mobility classification	True
B	ZSU 23-4 is Self-Propelled (SP)	True
C	ZSU 23-4 is Tracked	True

Table 11. Backwards Chaining Example.

Forwards and Backwards Chaining Discussion. From our examples of forwards and backwards chaining we can begin to see some potential advantages and disadvantages of both. According to Marakas there are two primary factors to consider when choosing whether to implement a forwards or backwards chaining inference mechanism. First and foremost the expert reasoning mode of the domain should be considered (Marakas, 1999, 244). If the experts of a domain reason similar to forward chaining then this may well be the determining factor. Choosing an inference mechanism with a reasoning procedure similar to the experts of a domain may serve to avoid problems with both Knowledge Representation and Knowledge Acquisition associated with reasoning. The other consideration is efficiency. If you have a large KB with a large number of goals in comparison to the amount of data or new facts, then forward chaining may be the most efficient (Marakas, 1999, 244). Backwards chaining in this case would be required to cycle through all the facts of the KB and match each fact to the goals it supports. Depending on the size of the KB it could take time.

Both methods can be counted on to establish a formal line of reasoning and arrive at satisfactory conclusion. The application should largely determine the procedure more suitable for inclusion.

Classification and Subsumption. Classification and subsumption are the main inference tasks for Description Logic (DL) based applications. Classification is simply checking if an object or concept belongs to a certain category (Russell, 1995, 323). Subsumption is related to classification in that it determines if one category is a subset of another based on the definitions and descriptions in the ontology (Russell, 1995, 323). As we know from our previous discussion the Web Ontology Language (OWL) will be named the W3C recommendation soon and it is a DL based language. Since OWL is DL based and is likely to be widely used and adopted throughout the SWEB, it is also very likely that many applications will employ classification and subsumption inference procedures. While First Order Logic (FOL) makes it easy to say things about objects, DLs provide a sophisticated system for defining categories of objects in terms of existing relations (Russell, 1995, 323). Of significant importance to these types of inference procedures are the concepts of inheritance and multiple inheritance. Inheritance for the purposes of this work is defined as the relationship among classes or concepts wherein one class or concept shares the structure, behavior or attributes defined in one or more other classes or concepts (Booch, 2001, 112). We refer to the class or concepts from which another class inherits as its superclass (Booch, 2001, 112). If the class or concept inherits from more than one class then we refer to this as multiple inheritance. Let us now turn to a live inference example to solidify our conceptual points.

d. Reasoning Examples

To demonstrate a small, simple aspect of the reasoning capabilities available to a semantically enabled system, let us implement the Fast Classifications of Terminologies (FaCT) Reasoner[101] against our Threat Antiaircraft Systems example. To

[101] FaCT was chosen for demonstration purposes because OilED provides an integrated reasoning environment without programmatic manipulation.

do so we will establish a supporting KB[102] inside the OilEd Version 3.5[103] (DIG) Ontology Development Environment (ODE). It is worth mentioning the reasoning and inference tools and engines currently available for use in semantically enabled applications are largely academic and austere[104]. However; significant, rapid improvements continue to be made. Our reasoning exemplar will employ the FaCT Reasoner capable of checking the satisfiability of a model by the class hierarchy and the discovering implied subsumptions in the model (Bechhofer, 2001).

(1) FaCT Reasoner. FaCT is a Description Logic (DL) based reasoner offering sound and complete reasoning supporting two DLs, SHQ or SHF[105] ([2] Bechhoffer, 2001, 7). FaCT reasoning is user executed by connecting to the FaCT reasoner and requesting verification of the model. FaCTs connection is through a CORBA based client server either running locally or remotely ([2] Bechhofer, 2001, 8).

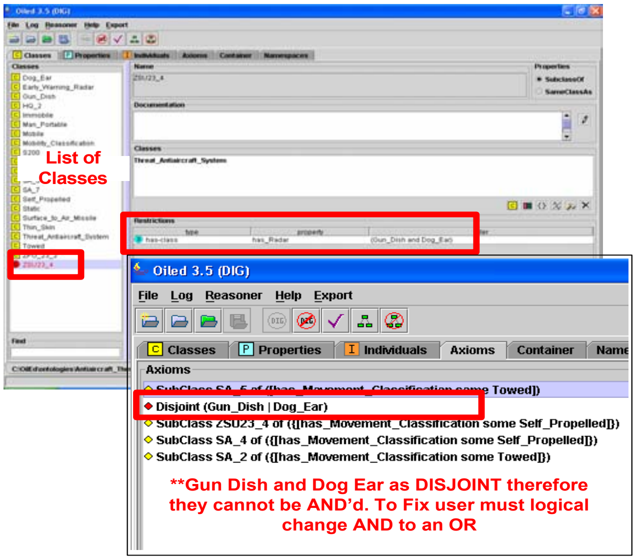
Satisfiability. When a user requests verification the ontology is translated into a SHQ knowledge base and sent to the reasoner for classification. Each class is checked for satisfiability by first determining the superclass of each class. After verification unsatisfiable concepts appear in red in the list of classes (See Figure 62). In the case below the class ZSU 23-4 is unsatisfiable due to the Gun Dish and Dog Ear radars being improperly AND’d when an axiom declares them as DISJOINT entities. The FaCT Reasoner catches the inconsistency and prompts the user to correct the error. A reasoning function in a SWEB application would prove valuable for logical validation of domain models and instances.

[102] KB defined as Ontology instantiated with instances (Noy and McGuinness).

[103] OilEd V 3.5 (DIG) was released November 2002 by the University of Manchester and Sean Bechhofer and Gary Ng. OilEd is an integrated, visual ontology editor capable of exporting an ontology in DAML, OWL, Dotty, and HTML presentation container.

[104] Some of the better developed reasoning systems include Java Expert System Shell (JESS) available from Sandia National Labratories, Java Theorem Prover (JTP) available from Knowledge Systems Laboratory at Stanford University, Fast Classification of Terminologies (FaCT) available from the University of Manchester.

[105] Current version runs SHQ DL. SHQ and SHF are both members of the DL family with their own language specifications.


OilEd 3.5 (DIG)
File Log Reasoner Help Export
Classes Properties Individuals Axioms Container Namespaces
Classes
Name
Properties
● SubclassOf
○ SameClassAs
Documentation
Classes
Restrictions
type
property
List of Classes
OilEd 3.5 (DIG)
File Log Reasoner Help Export
Classes Properties Individuals Axioms Container Name
Axioms

Dog_Ear
Early_Warning_Radar
Gun_Dish
HQ_2
Immobile
Man_Portable
Mobile
Mobility_Classification
S200
SA_7
Self_Propelled
Static
Surface_to_Air_Missile
Thin_Skin
Threat_Antiaircraft_System
Towed
ZSU23_4

Name: ZSU23_4

Classes: Threat_Antiaircraft_System

Restrictions: has-class has_Radar (Gun_Dish and Dog_Ear)

Axioms:

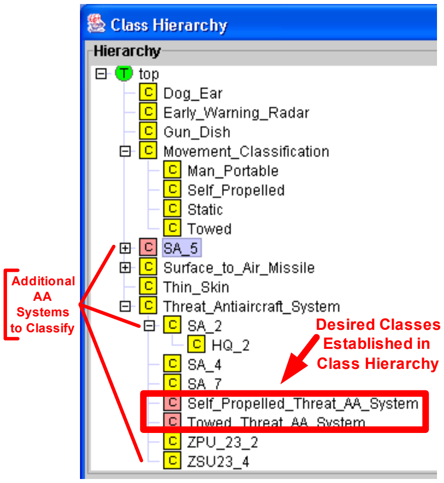
- SubClass SA_6 of ({[has_Movement_Classification some Towed])
- Disjoint (Gun_Dish | Dog_Ear)
- SubClass ZSU23_4 of ({[has_Movement_Classification some Self_Propelled]})
- SubClass SA_4 of ({[has_Movement_Classification some Self_Propelled]})
- SubClass SA_2 of ({[has_Movement_Classification some Towed]})

**Gun Dish and Dog Ear as DISJOINT therefore they cannot be AND'd. To Fix user must logical change AND to an OR

Figure 62. Unsatisfiable Example.

Subsumption Checking and Classification. Similar to the satisfiability function, when the user requests verification the reasoner looks to discover implicit subsumptions within the model. Once implicit subsumptions are located the hierarchy view is changed to reflect the newly discovered subsumptions. The user is again prompted to commit the newly identified subsumptions to the model or discard the results. Let us look at a live example of the subsumption checking and classification example with our Threat Antiaircraft Systems example.

The Implicit Subsumption in Threat Antiaircraft Mobility Classification. For our example let us take the Threat Antiaircraft (AA) Systems we have in our KB and further classify them. For our purposes we are interested in establishing the mobility classifications within our model. Currently our model has all the AA systems classified under the superclass Threat Antiaircraft Systems. To begin, let us further classify the Threat Antiaircraft Systems contained in our model into the more specific Mobility Classifications of Towed and Self Propelled. To start, we will establish the classes we wish our reasoner to classify our systems into. For our example the target classes are Self Propelled Threat AA System and Towed Threat AA System. Notice we use the naming convention for human readability and traceability back to its superclass just as OO design practices suggest. Now that we have established our classes shown in red in Figure 63, let us set the expressions and axioms necessary to orchestrate our classification. Figure 63 illustrates a form of the class hierarchy prior to invoking the reasoner. Notice the classes we established in preparation for reasoning operations.



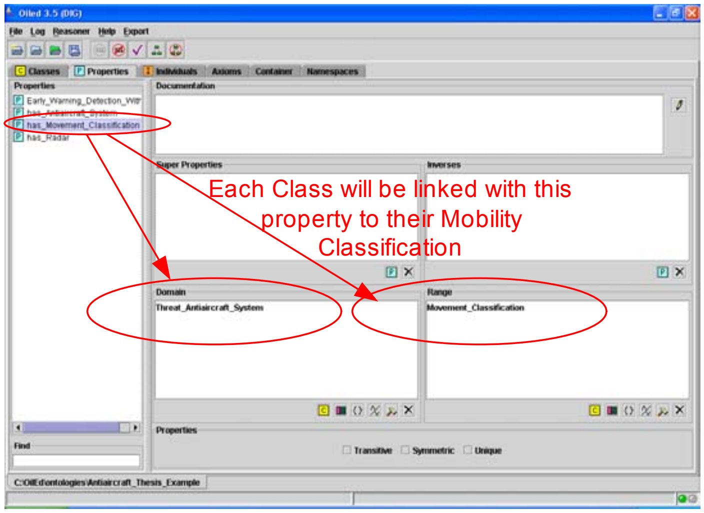
```
top
    Dog_Ear
    Early_Warning_Radar
    Gun_Dish
    Movement_Classification
        Man_Portable
        Self_Propelled
        Static
        Towed
    SA_5
    Surface_to_Air_Missile
    Thin_Skin
    Threat_Antiaircraft_System
        SA_2
            HQ_2
        SA_4
        SA_7
        Self_Propelled_Threat_AA_System
        Towed_Threat_AA_System
        ZPU_23_2
        ZSU23_4
```

Additional AA Systems to Classify

Desired Classes Established in Class Hierarchy

Figure 63. Established New Classes for FaCT Classification.

Next we will establish the property hasMovement Classification which we will include as a property restriction for every system we wish to classify. The domain will be set to the root class, Threat Antiaircraft Systems and the range set to the class Movement Classification (See Figure 64).



Properties

- Early_Warning_Detection_Wit
- has_Antiaircraft_System
- has_Movement_Classification
- has_Radar

Domain: Threat_Antiaircraft_System

Range: Movement_Classification

Properties: □ Transitive □ Symmetric □ Unique

C:\OilEd\ontologies\Antiaircraft_Thesis_Example

Each Class will be linked with this property to their Mobility Classification

Figure 64. Movement Classification Property.

Once the property is established we now have the conditions set to formulate the axioms that will classify our systems. When we establish the axiom we will establish the axiom for each system individually by asserting a subclass Axiom. For the ZSU 23-4 the subclass axiom takes the form of:

SubClass ZSU 23_4({[has_Movement_Classification some Self_Propelled]})

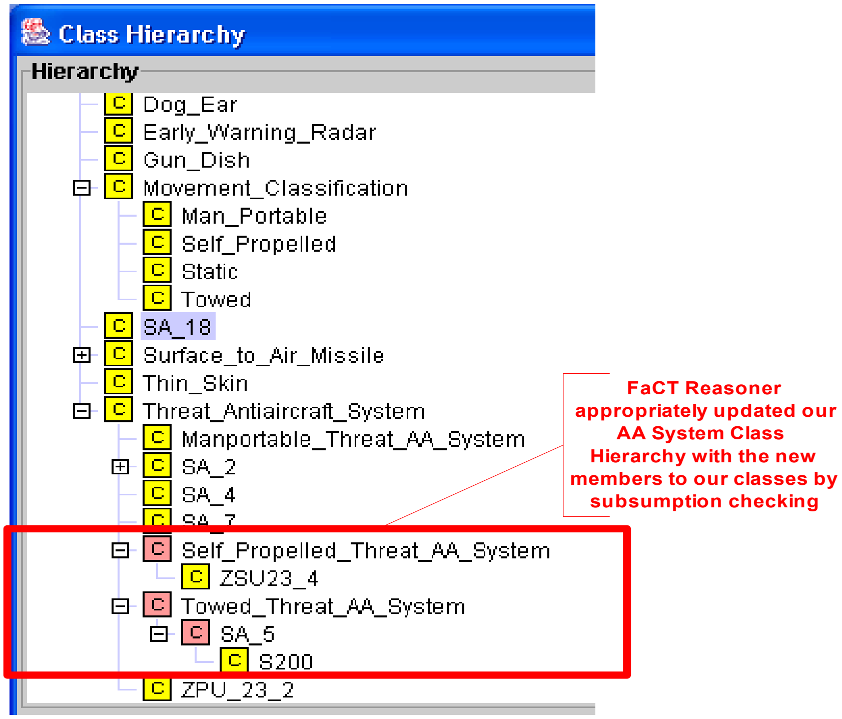
The axiom states the class ZSU23_4 has a movement classification of self propelled and should be classified in the Self_Propelled_Threat_AA_System Class. To offer provenance the Self_Propelled_Threat_AA_System Class requires all AA Systems with a Self Propelled movement classification to be members of its class extension.

Similar to the Self Propelled systems we created axioms for, the AA systems with a Towed Movement Classification must have their hidden subsumptions discovered in order to classify the towed systems into the Towed Threat AA System Class. The axiom for the SA 5 takes the form of:

SubClass SA_5({[has_Movement_Classification some Towed]})

Once the axioms are complete for our two example systems let us connect to the FaCT Reasoner to see our results.

We observe in Figure 65 that the FaCT Reasoner added ZSU 23-4 and SA 5 to our class hierarchy in their new classes without our model explicitly stating it. The FaCT reasoner discovered the implicit subsumption in the model and returned the results.

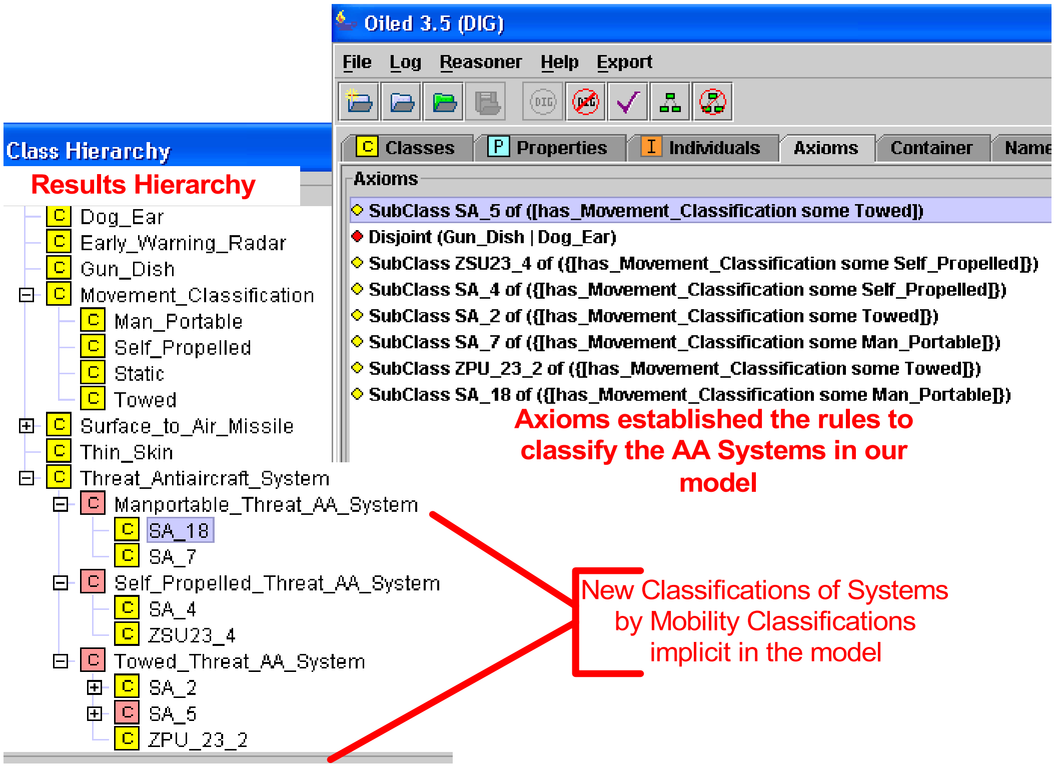


FaCT Reasoner appropriately updated our AA System Class Hierarchy with the new members to our classes by subsumption checking

Figure 65. FaCT Reasoner Results from Self Propelled and Towed Axioms.

This type of reasoning procedure can supply added value to SWEB applications by allowing data driven classification. In our example we imagine Intelligence Analysts now being spared the menial tasks of associating characteristics with a piece of threat equipment. The ontology, KB and Reasoner can classify the intelligence reports as they flow in. Granted an application such as the one we are speaking of will take a combined effort of all the concepts we have discussed thus far in this work, but it is not too far out of sight.[106] To demonstrate further value let us continue to establish axioms to cover all our systems, and further establish an even more granular classification on our new classes (See Figure 66).

[106] Following the successful implementation of the most basic inference capabilities, sophistication will increase and the complexity of the corresponding inferences will similarly increase.



Axioms

- SubClass SA_5 of ([has_Movement_Classification some Towed])
- Disjoint (Gun_Dish | Dog_Ear)
- SubClass ZSU23_4 of ({[has_Movement_Classification some Self_Propelled]})
- SubClass SA_4 of ({[has_Movement_Classification some Self_Propelled]})
- SubClass SA_2 of ({[has_Movement_Classification some Towed]})
- SubClass SA_7 of ({[has_Movement_Classification some Man_Portable]})
- SubClass ZPU_23_2 of ({[has_Movement_Classification some Towed]})
- SubClass SA_18 of ({[has_Movement_Classification some Man_Portable]})

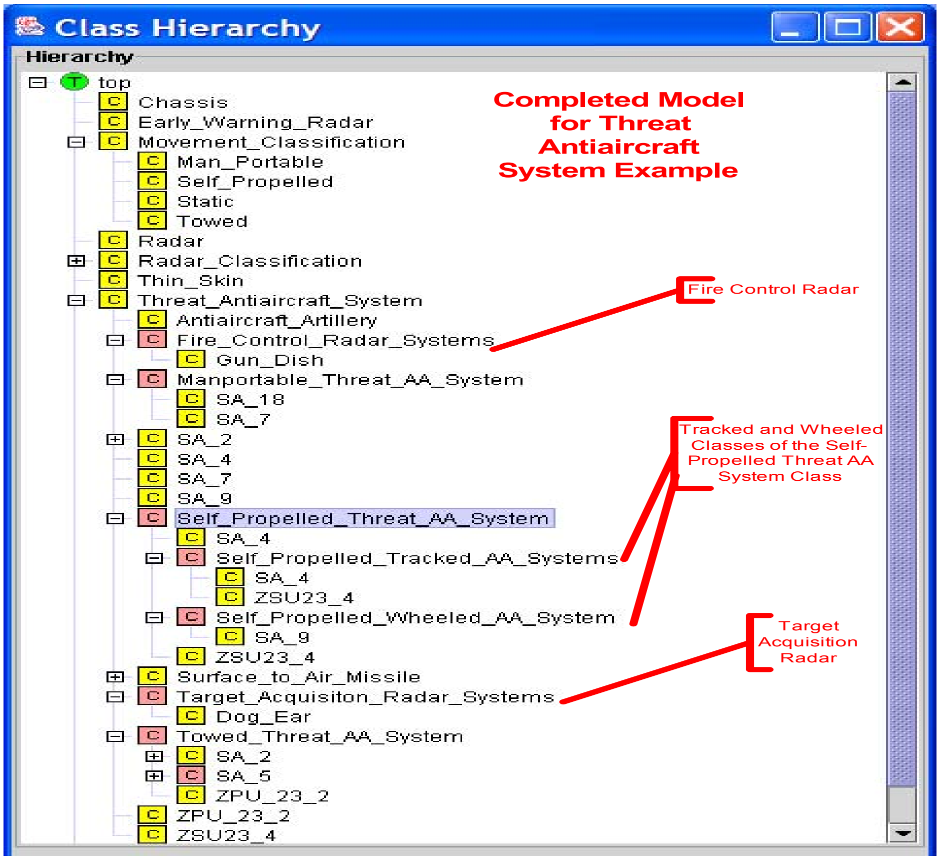
Results Hierarchy

Axioms established the rules to classify the AA Systems in our model

New Classifications of Systems by Mobility Classifications implicit in the model

Figure 66. KB Mobility Classification FaCT Reasoner Results.

Let us now further classify all systems in our Self Propelled Threat AA System by whether or not they are tracked or wheeled systems. While we set the conditions for this procedure, and to demonstrate further value, we will set up axioms to implicitly classify the radar systems we have in our KB into either the Target Acquisition or Fire Control Systems class. This will more thoroughly demonstrate how we can further classify other useful aspects of our KB and complete our example. It is important reiterate that these classifications were not explicitly stated in our model but were discovered by the FaCT Reasoner during subsumption checking. We will not describe all the details of this exemplar, but suffice it to say the conditions were set for these classifications just as our previous example. Our completed KB class hierarchy with subsumption additions for the Radar Classification and the Tracked and Wheeled classifications is illustrated in Figure 67.



```
top
  Chassis
  Early_Warning_Radar
  Movement_Classification
    Man_Portable
    Self_Propelled
    Static
    Towed
  Radar
  Radar_Classification
  Thin_Skin
  Threat_Antiaircraft_System
    Antiaircraft_Artillery
    Fire_Control_Radar_Systems
      Gun_Dish
    Manportable_Threat_AA_System
      SA_18
      SA_7
    SA_2
    SA_4
    SA_7
    SA_9
    Self_Propelled_Threat_AA_System
      SA_4
      Self_Propelled_Tracked_AA_Systems
        SA_4
        ZSU23_4
      Self_Propelled_Wheeled_AA_System
        SA_9
      ZSU23_4
    Surface_to_Air_Missile
    Target_Acquisiton_Radar_Systems
      Dog_Ear
    Towed_Threat_AA_System
      SA_2
      SA_5
      ZPU_23_2
    ZPU_23_2
    ZSU23_4
```

Completed Model for Threat Antiaircraft System Example

Fire Control Radar

Tracked and Wheeled Classes of the Self-Propelled Threat AA System Class

Target Acquisition Radar

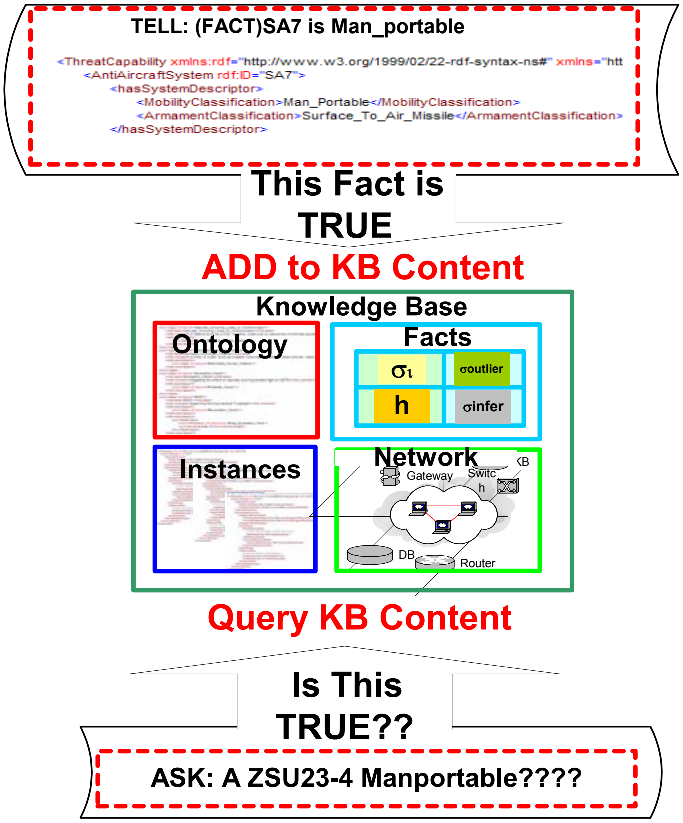
Figure 67. Completed Model Using FaCT Reasoner.

e. Functions

A KB must be designed to interact with human readers, inference procedures or agents (Russell, 1995, 218). It can also be the case that any given KB can interact with all of the potential actors, but must at least interact with one. To foster interaction the KB must have functions or operations allowing actors to manipulate content. The first two functions we will discuss are closely related; the first is TELL and the second is ASK.

(1) TELL. The TELL function of a KB allows a human, agent or inference procedure to inform the KB that a sentence is true (Levesque, 2000, 13). This function is essentially an assertion and serves as a mechanism by which to add facts to a KB. The KB and its embedded logic can either reply that a given TELL function is logically consistent with its beliefs or is not. If not, the facts associated with the TELL can be discarded or reformed.

(2) ASK. The ASK function asks the KB if a sentence is true (Levesque, 2000, 13). The ASK function is similar to a query. When an ASK function occurs the KB must determine if the proposition or facts within the ASK are in fact known. The result from an ASK function should be a simple yes or no. For a visual illustration of a TELL/ASK Function refer to Figure 68.



```
<ThreatCapability xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns="htt
    <AntiAircraftSystem rdf:ID="SA7">
        <hasSystemDescriptor>
            <MobilityClassification>Man_Portable</MobilityClassification>
            <ArmamentClassification>Surface_To_Air_Missile</ArmamentClassification>
        </hasSystemDescriptor>
```

Figure 68. TELL/ASK Functions.

f. Traceability

The traceability of information or pedigree is a necessary function of a KB in order for an acceptable level of trust to be established. Traceability returns output resembling a proof tree explicitly stating the fats and propositions that satisfy certain properties along with their answers (Fikes, 2003, 3). The traceability function is essentially the explanation capability of the KB that must be communicated in the user's terms, and above all be understandable. The traceability function of a KB allows the KB to follow the life of a proposition from the time it is told to the KB from its originating source to its usage. This aspect of traceability offers the KB a potential to learn in what circumstances certain propositions are used and to predict usage patterns in the future. Essentially, traceability can be a learning enabler.

g. Maintenance

The maintenance of a KB in a SWEB application must be simplified to the point it can be done by the domain experts. For the SWEB to be widely adopted this aspect of KB is one of many aspects that must be simplified.[107] A simplified maintenance scheme will be contrary to the KBs of ES which were painstakingly maintained by a cadre of knowledge engineers. The SWEB must ensure the maintenance of many different knowledge sources and subject matter experts' work can be maintained with relative simplicity, and with only the knowledge of the domain in which the KB serves. It is imperative the maintainers of the SWEB KBs be able to maintain a KB without formal training in Artificial Intelligence and still do an effective job.

5. External Rule Based System

While formally not considered a component of a KB, an external Rule Based System capable of interacting with the KB can provide a valuable function to an SWEB application. A Rule Based program in general is considered a declarative program describing what a computer should do if certain conditions exist, but allows another runtime program (execution engine) to determine how the computer should do it. Declarative programming is very different from the traditional procedural programming where the computer is told explicitly how to accomplish the task (Friedman-Hill, 2003, 16). For our discussion we will focus on the implementation of an external Rule Based System as a technique to avoid the dangers of establishing too rigid of a domain theory by transferring the application of our most specific, domain rules to the Rule Based System (Jess™). With this transfer we are building safeguards in to our application to help prevent our most specific rules, applicable only to our application area, from embedding in other aspects of our system, such as ontologies, agents, and knowledge stores that could create a tight domain theory discouraging widespread reuse and sharing

[107] As we indicated in the Ontology chapter, it is imperative that ontology development tools be widely available and easy to use for large-scale implementation of the SWEB.

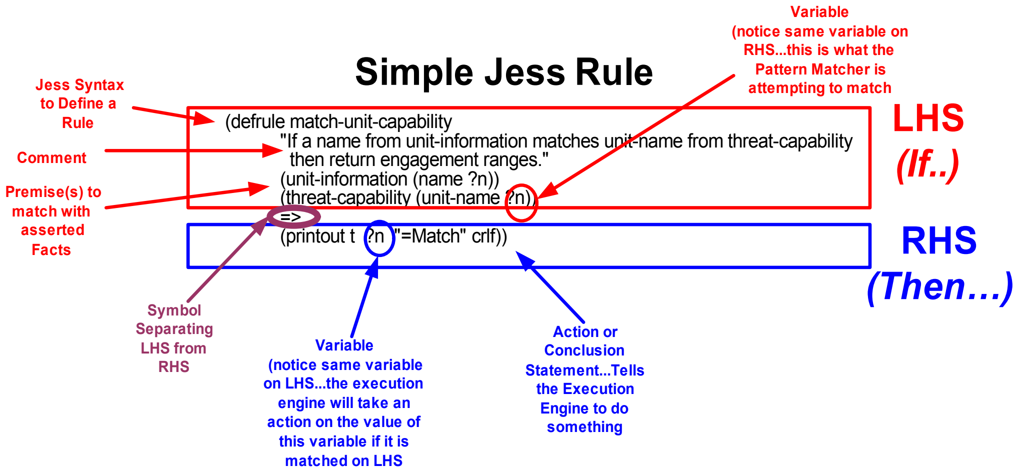
(interoperability). As we recall from our previous discussions in Chapter VI, domain theories should make as few claims about the domain as possible allowing the domain theory to remain flexible and easily extendable by other users and developers (Gruber, 1993, 3). In a sense, an external Rule Based System assists us in achieving this flexibility by separating our highly restrictive claims about the domain and externally applying it through rules on demand without effects on our more general domain theory. While there are many Rule Based Systems and Rule Engines[108] we will focus our discussion on the use of the Java™ Expert System Shell (Jess™) due to its active user group support, quality documentation and its ability to interoperate with Java™. While our discussion will be Jess™specific, many of the principles and theories can be applied to most other Rule Based Systems.

a. Rules

A Rule Based System uses rules to derive conclusions from premises (Friedman-Hill, 2003, 17). A rule can be viewed as a type of instruction organized with a premise on the Left-Hand-Side (LHS) of the equation, and an action on the Right-Hand-Side (RHS) (Friedman-Hill, 2003, 17). Rules can be likened to *If…Then* statements in which the LHS contains the *If,* or the test conditions, and the RHS contains the *Then*, or the actions. If all there is to Jess™ is that it simply functions as a series of *If…Then* statements, then why not just programmatically embed the *If…Then* statements. As will be demonstrated later, Jess™ uses the Rete algorithm, a technique for fast pattern matching enabling it to function orders of magnitude faster than *If…Then* statements (Friedman-Hill, 2003, 134), as well as enabling us to keep our domain theory loose. Figure 69 illustrates an example of a Jess™ rule we will apply in our working example later in this section. From Figure 69, we observe that a Jess™ rule has a recognizable syntax. Jess™ rules must conform to the constructs of the syntax to be validated and accepted into working memory. We will not cover the Jess™ syntax in detail in this discussion, with the exception of what we include in our example[109].

[108] See www.volantec.biz/rules.htm for a complete listing of available Rule Based Systems.

[109] For more information on the Jess™ syntax see the Jess™ 6.0 User Manual at [http://herzberg.ca.sandia.gov/jess/] or Rule-Based Systems in Java: Jess in Action by Ernest Friedman-Hill.


Simple Jess Rule
Jess Syntax to Define a Rule
Comment
Premise(s) to match with asserted Facts
Variable (notice same variable on RHS...this is what the Pattern Matcher is attempting to match
LHS
(If..)
RHS
(Then...)
Symbol Separating LHS from RHS
Variable (notice same variable on LHS...the execution engine will take an action on the value of this variable if it is matched on LHS
Action or Conclusion Statement...Tells the Execution Engine to do something

```
(defrule match-unit-capability
    "If a name from unit-information matches unit-name from threat-capability
     then return engagement ranges."
    (unit-information (name ?n))
    (threat-capability (unit-name ?n))
    =>
    (printout t ?n "=Match" crlf))
```

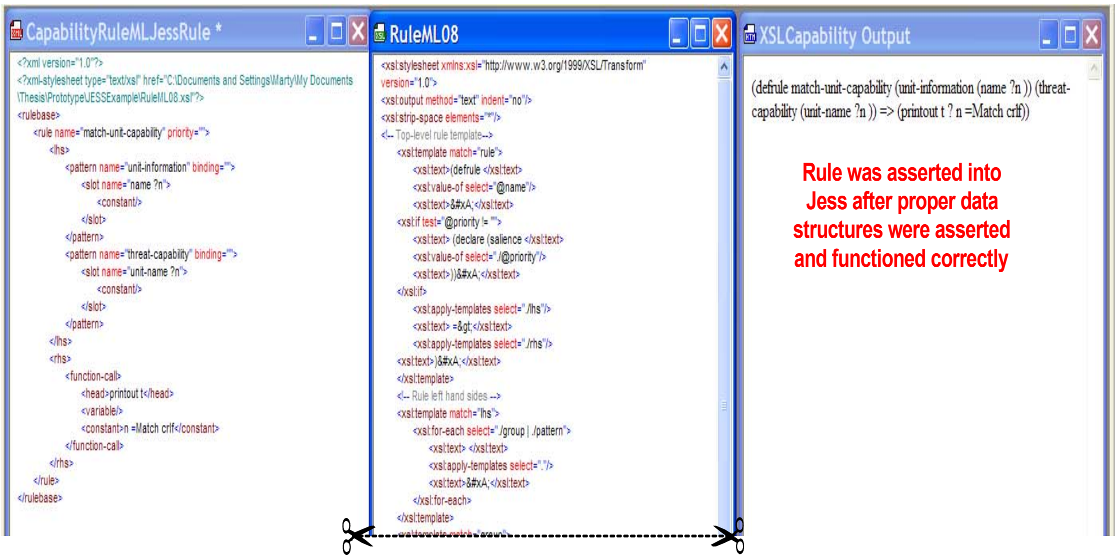
Figure 69. Anatomy of a Simple Jess™ Rule.

(1) XML and Jess™ Rules. Now that we are familiar with the importance and benefits of XML to the SWEB and interoperability in general, we would be remiss if we did not address how XML can be used in conjunction with Jess™. From our previous discussions and examples we demonstrated how XML, and its semantic variants OWL and DAML, can used to markup facts or contents of a KB, but as we shall soon demonstrate Jess™ rules can also be stored in a special XML language called the Rule Markup Language (RuleML). Again, from Figure 69, we observe that the Jess™ syntax is a text based. The RuleML and DAML Rules programs are underway to standardize the storage of Jess™ rules in XML and DAML, and very likely OWL. We will demonstrate an example of the power of XML based rule storage by illustrating the relative simplicity involved in transforming a rule stored in RuleML back to the Jess™ syntax by applying a stylesheet or XSL. The added flexibility of storing Jess™ Rules in RuleML will enable Jess™ to exploit the extensibility, interoperability, and flexibility of the XML storage format enhancing storage options, reuse and sharing (Freidman-Hill, 2003, 373)[110]. From RuleML, as we will demonstrate, a Jess™ rule can be easily transformed to proper Jess™ syntax in preparation for assertion into Jess™ working

[110] For more information on RuleML see [http://dfki.uni-kl.de/ruleml/].

memory. The DAML and OWL efforts will further add a self describing nature as well as adding meaning and context to the rules, making them ready for immediate inclusion into the SWEB.

Another option offered to Jess™ rules by the XML storage format is the ability to use Java™ and DOM or JDOM to read in the XML enabled Jess™ rule document and programmatically assert the rule in working memory through Java™ within the Jess™ environment. Figure 70 is an example of the Jess™ rule we used in Figure 69 marked up in RuleML and its XSL transformation[111] to proper Jess™ syntax (Friedman-Hill, 2003, 370). Standardized XML versions of Jess™ rules and data structures are critical to extending the capabilities and implementation options of Jess™ within the SWEB environment.



RuleML Version on match-unit-capability

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="C:\Documents and Settings\Marty\My Documents
\Thesis\Prototype\JESSExample\RuleML08.xsl"?>
<rulebase>
    <rule name="match-unit-capability" priority="">
        <lhs>
            <pattern name="unit-information" binding="">
                <slot name="name ?n">
                    <constant/>
                </slot>
            </pattern>
            <pattern name="threat-capability" binding="">
                <slot name="unit-name ?n">
                    <constant/>
                </slot>
            </pattern>
        </lhs>
        <rhs>
            <function-call>
                <head>printout t</head>
                <variable/>
                <constant>n =Match crlf</constant>
            </function-call>
        </rhs>
    </rule>
</rulebase>
```

XSL from Jess in Action Source Code

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="text" indent="no"/>
<xsl:strip-space elements="*"/>
<!-- Top-level rule template-->
    <xsl:template match="rule">
        <xsl:text>(defrule </xsl:text>
        <xsl:value-of select="@name"/>
        <xsl:text>&#xA;</xsl:text>
    <xsl:if test="@priority != ''">
        <xsl:text> (declare (salience </xsl:text>
        <xsl:value-of select="./@priority"/>
        <xsl:text>))&#xA;</xsl:text>
    </xsl:if>
        <xsl:apply-templates select="./lhs"/>
        <xsl:text> =&gt;</xsl:text>
        <xsl:apply-templates select="./rhs"/>
    <xsl:text>)&#xA;</xsl:text>
    </xsl:template>
    <!-- Rule left hand sides -->
    <xsl:template match="lhs">
        <xsl:for-each select="./group | ./pattern">
            <xsl:text> </xsl:text>
            <xsl:apply-templates select="."/>
            <xsl:text>&#xA;</xsl:text>
        </xsl:for-each>
    </xsl:template>
```

**Snippet from actual XSL to illustrate transformation

Text Output onto proper Jess Syntax

```
(defrule match-unit-capability (unit-information (name ?n )) (threat-
capability (unit-name ?n )) => (printout t ? n =Match crlf))
```

Rule was asserted into Jess after proper data structures were asserted and functioned correctly

Figure 70. RuleML Version of a Jess™ Rule and Its XSL Transformation into Valid Jess™ Syntax.

[111] XSL available in Rule-Based Systems in Java: Jess in Action Source Code along with a validating DTD. XML interoperability is listed as an area to be addressed in the Jess™ Development Roadmap Survey results recently published on the Jess™ website.

b. Rule Based System Components

To further understand how a Rule Based System functions it is helpful to decompose the system to its basic components and analyze each component for its functionality. A Rule Based System can be decomposed to four components (Friedman-Hill, 2003, 19):

- Inference Engine (Consists of Pattern Matcher, Agenda and Execution Engine)
- Rule Base
- Working Memory

The functionality of the four components combined form the Rule Based System. To gain familiarity with each of these components refer to Figure 71 where we enumerate the contributions of each components to the overall Rule Based System with a graphical depiction. We will continue to explain other aspects of Figure 71 as we proceed through our working demonstration.

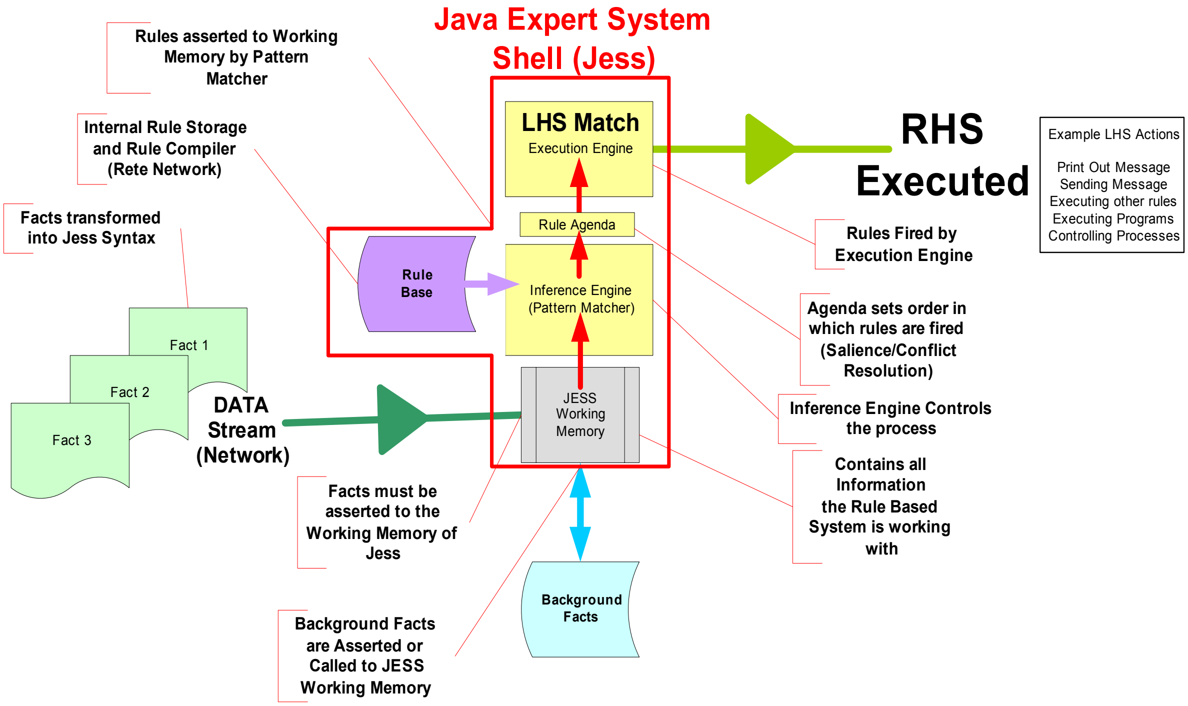

Java Expert System Shell (Jess)
Rules asserted to Working Memory by Pattern Matcher
Internal Rule Storage and Rule Compiler (Rete Network)
Facts transformed into Jess Syntax
LHS Match
Execution Engine
Rule Agenda
Rule Base
Inference Engine (Pattern Matcher)
JESS Working Memory
RHS Executed
Example LHS Actions
Print Out Message
Sending Message
Executing other rules
Executing Programs
Controlling Processes
Rules Fired by Execution Engine
Agenda sets order in which rules are fired (Salience/Conflict Resolution)
Inference Engine Controls the process
Contains all Information the Rule Based System is working with
Fact 1
Fact 2
Fact 3
DATA Stream (Network)
Facts must be asserted to the Working Memory of Jess
Background Facts
Background Facts are Asserted or Called to JESS Working Memory

Figure 71. Rule Based System Data Flow.

c. *Rule Based System Example with Jess™*

Before beginning with our example we must set the conditions required for our Jess™ example to properly function. According to Ernest Friedman-Hill, the inventor of Jess™, the process of developing a Rule Based System consists of the following six steps (Friedman-Hill, 2003, 26):

- Knowledge Engineering
- Establishing Data Structures for Facts
- Continuous Testing
- Building and Appropriate and Usable Interface
- Writing the Rules
- Iteration

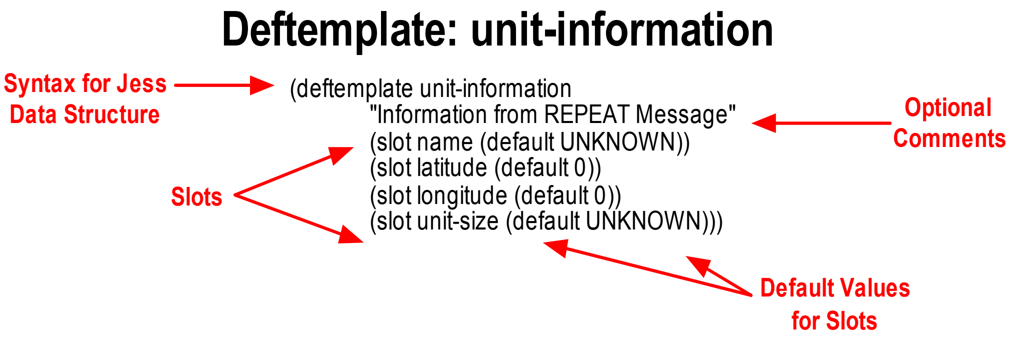
The six steps described above were used when developing our example with the exception of Step 4. Our example will be executed entirely from the command line as we eventually intend to embed Jess™ functionality within software agents (See Future Work). To begin let us describe the scenario for our example and discuss the preconditions and assumptions.

(1) Example Scenario. Our example will continue on with the Threat Antiaircraft System thread we began building in the Ontology Chapter. Our requirement is to implement an external Rule Based System using Jess™ into our notional SWEB application. Our Rule Based Jess™ System must be to interact with a data feed carrying messages containing enemy unit information. From the enemy unit information our system must be able to recognize the unit name and attempt to match that unit name to like unit name(s) in our Order of Battle Files (background facts). Once a match occurs, our system must then retrieve and associate additional background facts about the threat anti-aircraft system (engagement and detection ranges) and output the values to the terminal. The system must also recognize if matching facts do not exist and return a "No Match" message, as well as a recommendation to initiate intelligence collection on the unit not matching our background facts. While this is a simple example, it demonstrates five of the six development steps prescribed above as well as the basic functionality of Jess™.

Scenario Assumptions.

- We have a working SWEB application
- The Jess™ rules entered our system in RuleML and have already been transformed
- This scenario will be automated with agents in future work
- The knowledge engineering was completed by drawing on our previous example
- The background facts or the threat-capability facts were stored in our KB holdings and asserted to Jess™ working memory

Develop and Assert the Jess ™ Data structures. Since we already collected the necessary knowledge for our previous Threat Anti-Aircraft examples we are afforded the luxury of reusing it for inclusion in our Rule Based System and can move to Step 2, which is to develop and assert the data structures for our example. Since our requirement dictates our Rule Based System must recognize facts from a data stream, we must have knowledge of the content and structure of the messages within the data stream. We in fact know the data stream contains enemy unit information consisting of unit name, latitude, longitude, and unit size. These attributes will become the slots in our Jess™ data structures. Since we know what the data stream contains we can begin building our deftemplate for unit-information with the slots we previously identified. Slots can be likened to columns or attributes in a relational database. We will name the deftemplate unit-information, after its contents, and establish slots for name, latitude, longitude and unit-size. Within the deftemplate we can establish default values for the slots similar to a relational database. In our deftemplate we will in fact establish default values for the name and unit-size slots. Our data structure for unit-information is depicted in Figure 72.

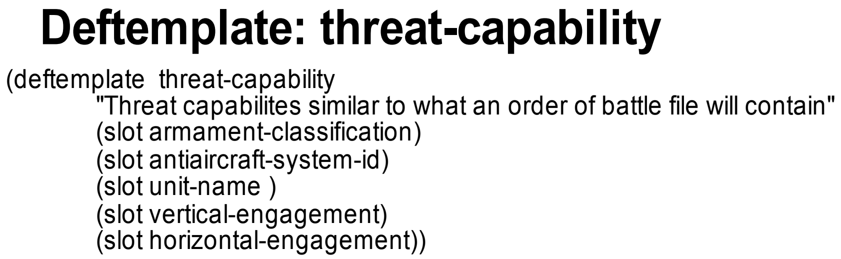


Deftemplate: unit-information

```
(deftemplate unit-information
        "Information from REPEAT Message"
        (slot name (default UNKNOWN))
        (slot latitude (default 0))
        (slot longitude (default 0))
        (slot unit-size (default UNKNOWN)))
```

Figure 72. Jess™ Deftemplate: Unit-Information.

Now that we have designed and established the deftemplate for the unit-information we must do the same for our background facts. To accomplish this we must establish the threat-capability deftemplate. Again, we must possess some knowledge of the structure of the source of the data. Since our internal KB possesses the threat-capability data (we own it), we can define our background facts deftemplate. Figure 73 depicts the threat-capability deftemplate.

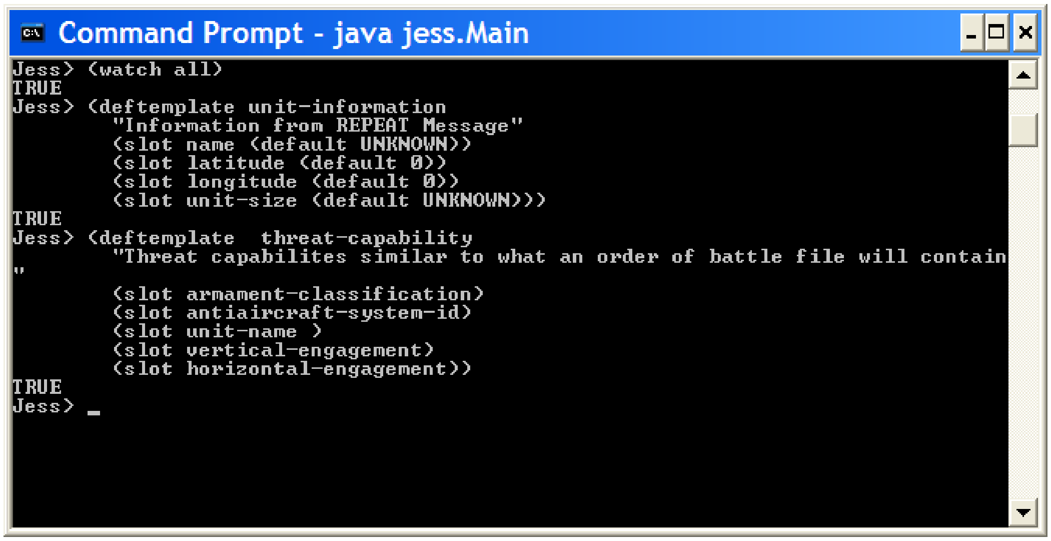


Deftemplate: threat-capability

```
(deftemplate  threat-capability
        "Threat capabilites similar to what an order of battle file will contain"
        (slot armament-classification)
        (slot antiaircraft-system-id)
        (slot unit-name )
        (slot vertical-engagement)
        (slot horizontal-engagement))
```

Figure 73. Jess™ Deftemplate: Threat-Capability.

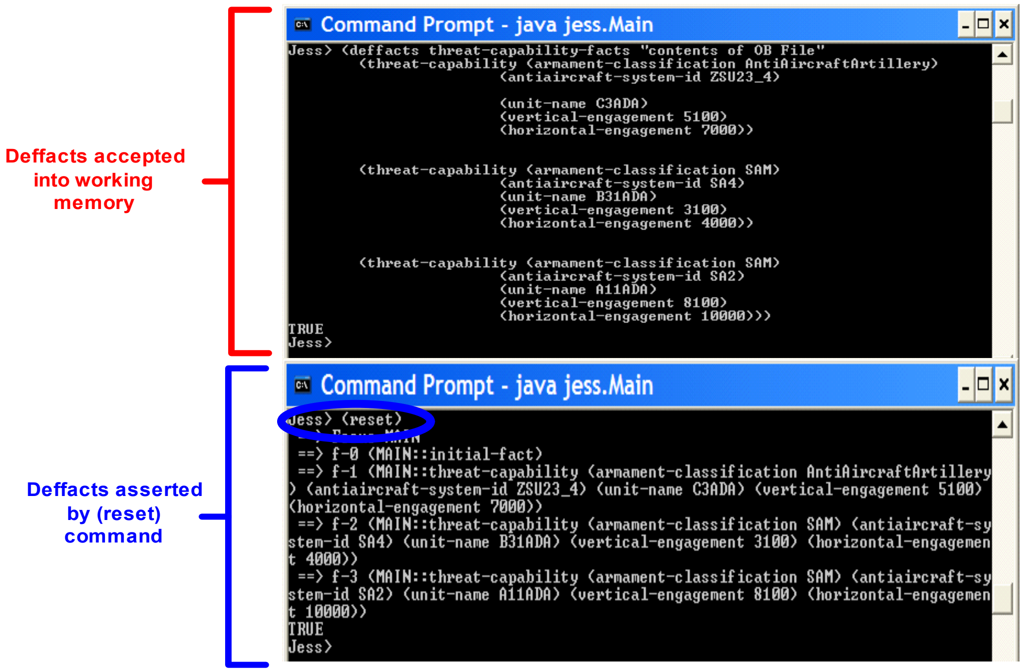
Assert deftempates to Jess ™ Working Memory. Now that we have our deftemplates designed, they are now ready to be asserted to working memory. To do this from the command line we simply copy the deftemplates from the text editor directly to the command line. Because Jess™ does not allow invalid syntax to be asserted to working memory, when we assert our deftemplates we should receive a `TRUE` return message from Jess™. Figure 74 is a screen capture of asserting both the unit-information and the threat-capability deftemplates to Jess™ working memory.



```
Jess> (watch all)
TRUE
Jess> (deftemplate unit-information
        "Information from REPEAT Message"
        (slot name (default UNKNOWN))
        (slot latitude (default 0))
        (slot longitude (default 0))
        (slot unit-size (default UNKNOWN)))
TRUE
Jess> (deftemplate  threat-capability
        "Threat capabilites similar to what an order of battle file will contain
"

        (slot armament-classification)
        (slot antiaircraft-system-id)
        (slot unit-name )
        (slot vertical-engagement)
        (slot horizontal-engagement))
TRUE
Jess> _
```

Figure 74. Assertion of Deftemplates to Working Memory.

Instantiate the Data Structures By Asserting the Facts. Our data structures are now in working memory ready to be instantiated with facts. Since the unit-information facts will be used to replicate the data stream we will assert them one at a time in order to view the activations and firing of rules. The background threat-capability facts can be asserted in bulk by defining the facts in a deffacts construct, or a named grouping of facts, and then invoking the `(reset)` command. Until the `(reset)` command is invoked, the deffacts, while resident in working memory are not yet asserted. Since our background facts are about threat-capabilities, we will name our deffacts construct threat-capability-facts. Note that our deffacts construct contains only the entities described by the data structures as validated by Jess™. This note will also apply to rules as we will discuss in the next section. Any deviation will result in an error message from Jess™. Therefore, facts must be asserted after the data structures, or deftemplates before deffacts. Figure 75 illustrates the threat-capability-facts as they are asserted into Jess™ working memory.

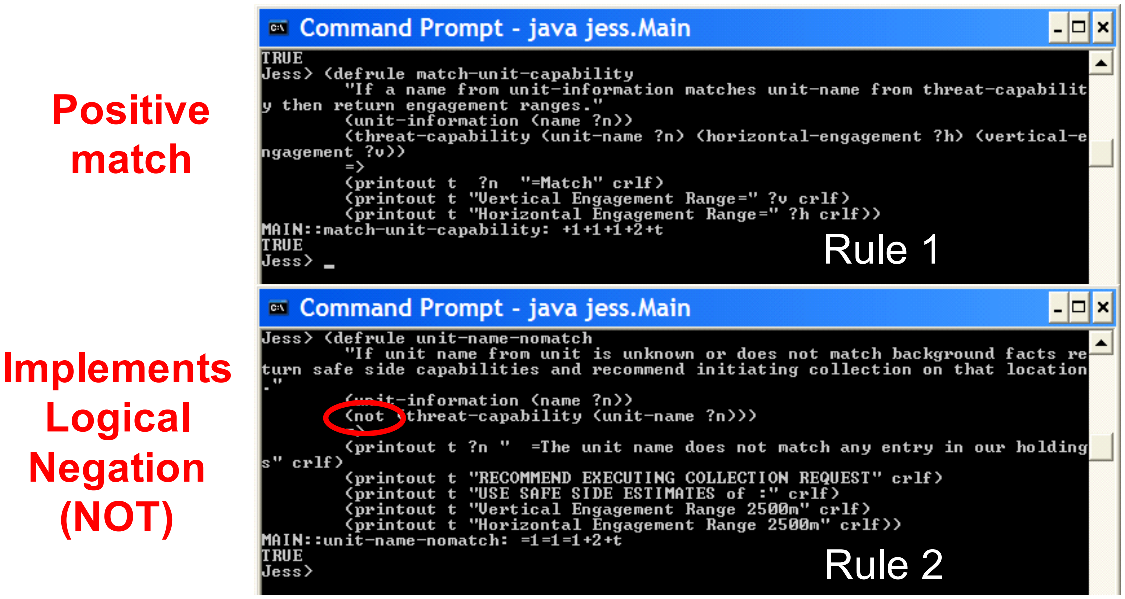

Command Prompt - java jess.Main
Deffacts accepted into working memory
Command Prompt - java jess.Main
Deffacts asserted by (reset) command

```
Jess> (deffacts threat-capability-facts "contents of OB File"
        (threat-capability (armament-classification AntiAircraftArtillery)
                           (antiaircraft-system-id ZSU23_4)

                           (unit-name C3ADA)
                           (vertical-engagement 5100)
                           (horizontal-engagement 7000))


        (threat-capability (armament-classification SAM)
                           (antiaircraft-system-id SA4)
                           (unit-name B31ADA)
                           (vertical-engagement 3100)
                           (horizontal-engagement 4000))


        (threat-capability (armament-classification SAM)
                           (antiaircraft-system-id SA2)
                           (unit-name A11ADA)
                           (vertical-engagement 8100)
                           (horizontal-engagement 10000)))
TRUE
Jess>
```

```
Jess> (reset)
 ==> Focus MAIN
 ==> f-0 (MAIN::initial-fact)
 ==> f-1 (MAIN::threat-capability (armament-classification AntiAircraftArtillery
) (antiaircraft-system-id ZSU23_4) (unit-name C3ADA) (vertical-engagement 5100)
(horizontal-engagement 7000))
 ==> f-2 (MAIN::threat-capability (armament-classification SAM) (antiaircraft-sy
stem-id SA4) (unit-name B31ADA) (vertical-engagement 3100) (horizontal-engagemen
t 4000))
 ==> f-3 (MAIN::threat-capability (armament-classification SAM) (antiaircraft-sy
stem-id SA2) (unit-name A11ADA) (vertical-engagement 8100) (horizontal-engagemen
t 10000))
TRUE
Jess>
```

Figure 75. Assertion of Deffacts to Jess™ Working Memory.

Establish and Assert the Rules. Since we have previously discussed the simple anatomy of a Jess™ rule from our discussion above (Figure 69), we have established adequate background knowledge to design and establish the rules for our Rule Based System. The first rule we will assert to our system is the rule we used in Figure 69 to illustrate the components of a rule. The match-unit capability rule attempts to find a match between the name slot of unit-information as represented by variable ?n and the unit-name slot in the threat-capability deffacts. If the match is successful, the RHS of the rule prints a message to the terminal/command line that the value of ?n= Match and the Vertical and Horizontal Engagement Ranges (?v and ?h) should also be retrieved from the matching unit in the threat-capability deffacts and the values printed to the terminal. Our second rule, the unit-name-nomatch rule is very similar in construct to our first rule except that it employs logical negation (NOT) to the LHS. By applying the NOT to the LHS of the rule the system will identify the names of all facts asserted that do not match the unit-name of any of our background facts. The RHS of the rule simply prints messages to the terminal warning the reader of “no match” and recommending a course of action to remedy the situation. Figure 76 illustrates the two rules as they are asserted into working memory.

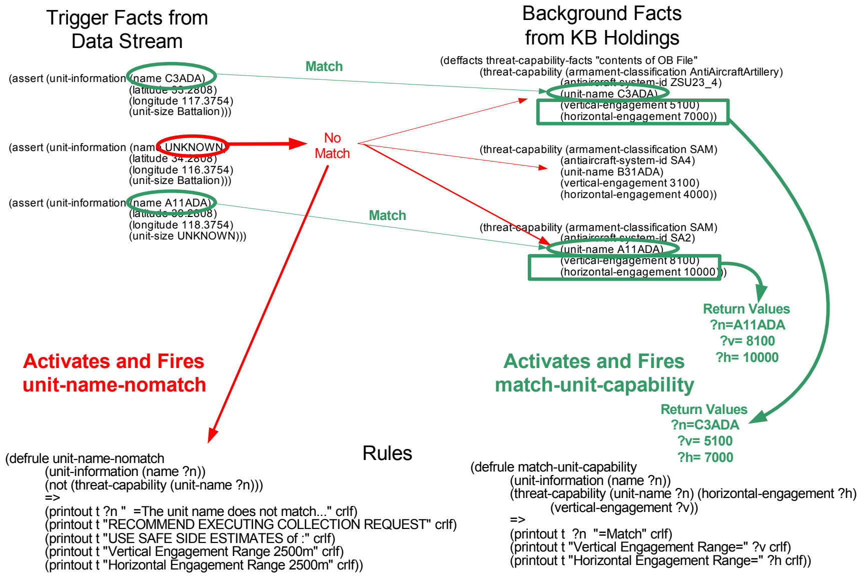

Positive match
Implements Logical Negation (NOT)
Command Prompt - java jess.Main
Rule 1
Command Prompt - java jess.Main
Rule 2

```
TRUE
Jess> (defrule match-unit-capability
        "If a name from unit-information matches unit-name from threat-capabilit
y then return engagement ranges."
        (unit-information (name ?n))
        (threat-capability (unit-name ?n) (horizontal-engagement ?h) (vertical-e
ngagement ?v))
        =>
        (printout t  ?n  "=Match" crlf)
        (printout t "Vertical Engagement Range=" ?v crlf)
        (printout t "Horizontal Engagement Range=" ?h crlf))
MAIN::match-unit-capability: +1+1+1+2+t
TRUE
Jess> _
```

```
Jess> (defrule unit-name-nomatch
        "If unit name from unit is unknown or does not match background facts re
turn safe side capabilities and recommend initiating collection on that location
."
        (unit-information (name ?n))
        (not (threat-capability (unit-name ?n)))
        =>
        (printout t ?n "  =The unit name does not match any entry in our holding
s" crlf)
        (printout t "RECOMMEND EXECUTING COLLECTION REQUEST" crlf)
        (printout t "USE SAFE SIDE ESTIMATES of :" crlf)
        (printout t "Vertical Engagement Range 2500m" crlf)
        (printout t "Horizontal Engagement Range 2500m" crlf))
MAIN::unit-name-nomatch: =1=1=1+2+t
TRUE
Jess>
```

Figure 76. Jess™ Rules Asserted.

Assert the Facts from the Data Stream. Now that the data structures (deftemplates), background facts (deffacts) and the rules (defrules) are asserted to Jess™ working memory, the conditions are set for Jess™ to run our example. To execute our example, we must assert the facts representing the data stream. Since we already asserted the data structure (deftemplate) for these facts we have a reasonable idea of what those facts might look like. Since these facts will be the mechanism activating/triggering our rules, we will assert them in isolation to observe the behavior of Jess™. To assert a fact simply preface the fact with the (`assert`) command and barring syntax errors, it will assert to working memory. Before we assert the rules and demonstrate the functionality of Jess™, let us look at the facts we will assert in order to pre-identify for the purposes of illustration which facts will activate which rule. Figure 77 identifies the actions for each specific rule.



```
(assert (unit-information (name C3ADA)
                          (latitude 33.2808)
                          (longitude 117.3754)
                          (unit-size Battalion)))

(assert (unit-information (name UNKNOWN)
                          (latitude 34.2808)
                          (longitude 116.3754)
                          (unit-size Battalion)))

(assert (unit-information (name A11ADA)
                          (latitude 30.2808)
                          (longitude 118.3754)
                          (unit-size UNKNOWN)))
```

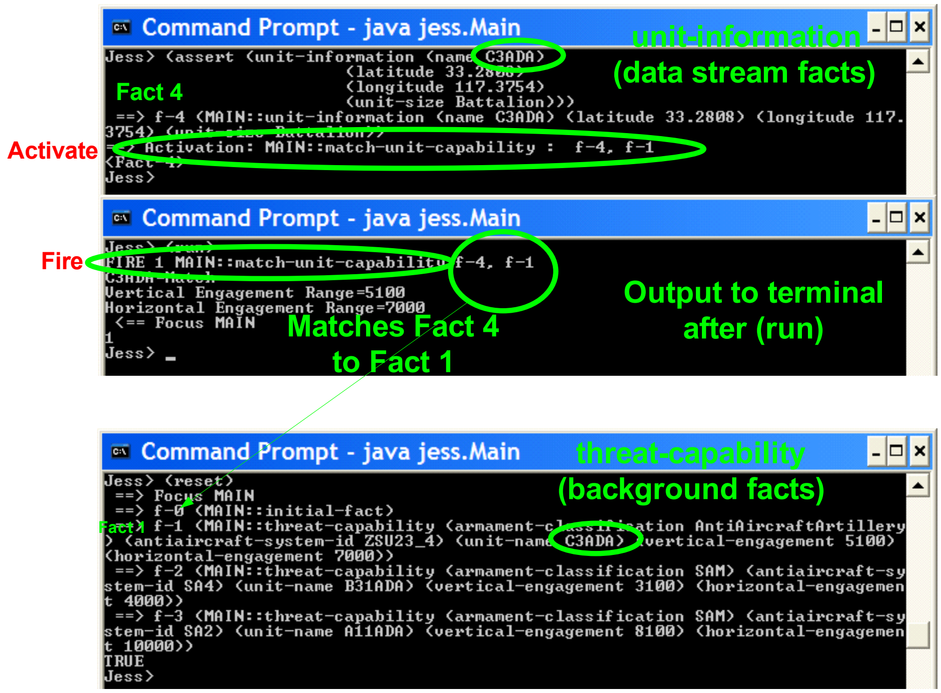
```
(deffacts threat-capability-facts "contents of OB File"
        (threat-capability (armament-classification AntiAircraftArtillery)
                (antiaircraft-system-id ZSU23_4)
                (unit-name C3ADA)
                (vertical-engagement 5100)
                (horizontal-engagement 7000))

        (threat-capability (armament-classification SAM)
                (antiaircraft-system-id SA4)
                (unit-name B31ADA)
                (vertical-engagement 3100)
                (horizontal-engagement 4000))

        (threat-capability (armament-classification SAM)
                (antiaircraft-system-id SA2)
                (unit-name A11ADA)
                (vertical-engagement 8100)
                (horizontal-engagement 10000)))
```

```
(defrule unit-name-nomatch
        (unit-information (name ?n))
        (not (threat-capability (unit-name ?n)))
        =>
        (printout t ?n "  =The unit name does not match..." crlf)
        (printout t "RECOMMEND EXECUTING COLLECTION REQUEST" crlf)
        (printout t "USE SAFE SIDE ESTIMATES of :" crlf)
        (printout t "Vertical Engagement Range 2500m" crlf)
        (printout t "Horizontal Engagement Range 2500m" crlf))
```

```
(defrule match-unit-capability
        (unit-information (name ?n))
        (threat-capability (unit-name ?n) (horizontal-engagement ?h)
                (vertical-engagement ?v))
        =>
        (printout t  ?n  "=Match" crlf)
        (printout t "Vertical Engagement Range=" ?v crlf)
        (printout t "Horizontal Engagement Range=" ?h crlf))
```

Figure 77. Rule Activations Associated with Facts.

Now that we know what we should expect we can now assert our trigger facts with some confidence (See Figure 78).



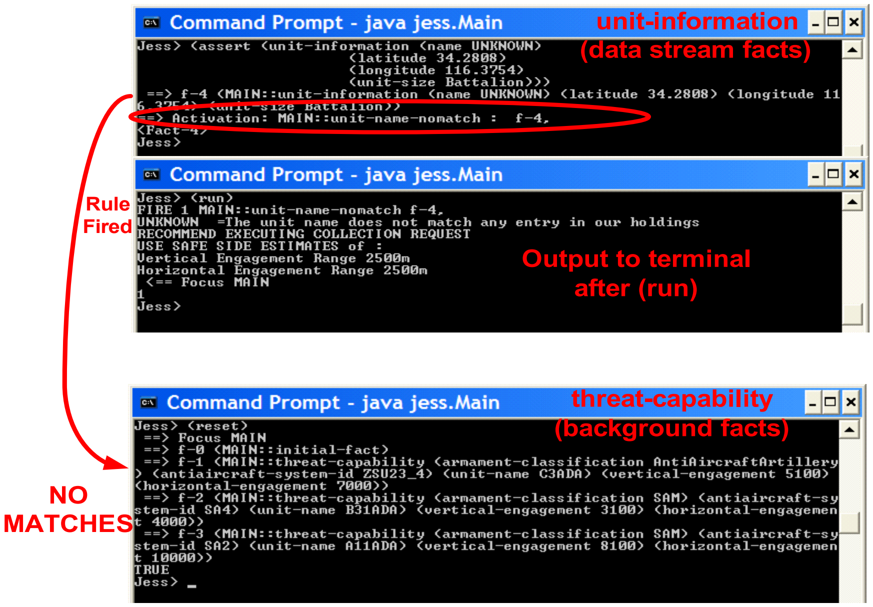
```
Jess> (assert (unit-information (name C3ADA)
                               (latitude 33.2808)
                               (longitude 117.3754)
                               (unit-size Battalion)))
 ==> f-4 (MAIN::unit-information (name C3ADA) (latitude 33.2808) (longitude 117.
3754) (unit-size Battalion))
==> Activation: MAIN::match-unit-capability :  f-4, f-1
<Fact-4>
Jess>
```

```
Jess> (run)
FIRE 1 MAIN::match-unit-capability f-4, f-1
C3ADA=Match
Vertical Engagement Range=5100
Horizontal Engagement Range=7000
 <== Focus MAIN
1
Jess> _
```

```
Jess> (reset)
 ==> Focus MAIN
 ==> f-0 (MAIN::initial-fact)
 ==> f-1 (MAIN::threat-capability (armament-classification AntiAircraftArtillery
) (antiaircraft-system-id ZSU23_4) (unit-name C3ADA) (vertical-engagement 5100)
(horizontal-engagement 7000))
 ==> f-2 (MAIN::threat-capability (armament-classification SAM) (antiaircraft-sy
stem-id SA4) (unit-name B31ADA) (vertical-engagement 3100) (horizontal-engagemen
t 4000))
 ==> f-3 (MAIN::threat-capability (armament-classification SAM) (antiaircraft-sy
stem-id SA2) (unit-name A11ADA) (vertical-engagement 8100) (horizontal-engagemen
t 10000))
TRUE
Jess>
```

Figure 78. Activation and Firing of Unit-Match-Capability Rule.

As expected, the unit-match-capability rule was fired once the C3ADA unit-information facts was asserted to working memory because it matched C3ADA in the threat-capability unit-name in background data. As we can observe in Figure 78 we are first alerted to the match by the rule activation and then after the rule fires, the LHS prints to the terminal the unit name along with horizontal and vertical engagement ranges. This system performed as we expected. Now that we have the conditions established for the unit-match capability rule to fire, let us look at the unit-name-nomatch rule as it is activated and fired (See Figure 79).



```
Jess> (assert (unit-information (name UNKNOWN)
                              (latitude 34.2808)
                              (longitude 116.3754)
                              (unit-size Battalion)))
 ==> f-4 (MAIN::unit-information (name UNKNOWN) (latitude 34.2808) (longitude 11
6.3754) (unit-size Battalion))
==> Activation: MAIN::unit-name-nomatch :  f-4,
<Fact-4>
Jess>
```

```
Jess> (run)
FIRE 1 MAIN::unit-name-nomatch f-4,
UNKNOWN  =The unit name does not match any entry in our holdings
RECOMMEND EXECUTING COLLECTION REQUEST
USE SAFE SIDE ESTIMATES of :
Vertical Engagement Range 2500m
Horizontal Engagement Range 2500m
 <== Focus MAIN
1
Jess>
```

```
Jess> (reset)
 ==> Focus MAIN
 ==> f-0 (MAIN::initial-fact)
 ==> f-1 (MAIN::threat-capability (armament-classification AntiAircraftArtillery
) (antiaircraft-system-id ZSU23_4) (unit-name C3ADA) (vertical-engagement 5100)
(horizontal-engagement 7000))
 ==> f-2 (MAIN::threat-capability (armament-classification SAM) (antiaircraft-sy
stem-id SA4) (unit-name B31ADA) (vertical-engagement 3100) (horizontal-engagemen
t 4000))
 ==> f-3 (MAIN::threat-capability (armament-classification SAM) (antiaircraft-sy
stem-id SA2) (unit-name A11ADA) (vertical-engagement 8100) (horizontal-engagemen
t 10000))
TRUE
Jess> _
```

Figure 79. Activation and Firing of Unit-Name-Nomatch Rule.

Again, as expected, the fact we asserted with a name (UNKNOWN) did not match any of our unit-names in our background fact holdings. As such, the unit-name-nomatch was activated and fired returning the desired output to the terminal.

How Jess™ works: the Rete Algorithm. As briefly mentioned above, Jess™ uses the Rete algorithm to execute fast pattern matching. The Rete works by constructing a network at runtime within the execution engine by representing the LHS test as nodes within a network (Friedman-Hill, 2003, 136). Each node (LHS test) can have one (tests one fact) or two inputs (tests across multiple facts) and unlimited numbers of outputs. As the facts are asserted to working memory they are processed through these nodes that form the Rete network until reaching the terminal node where the output is rendered. Once a fact has successfully navigated the nodes of the Rete network, meaning all the LHS conditions were met, the rules located in the terminal node may be applied to the facts (Friedman-Hill, 2003, 137). It is important to note the Rete network is formed in working memory at runtime. By doing this Jess™ essentially stores previous pattern matching results and is only required to fully process facts not matching the activated patterns of the facts previously asserted. In essence, it is as if Jess™ executes an *If…Then* statement, remembering the fact pattern that satisfied the *If* (LHS) conditions. The next time the same pattern is met Jess™ simply recalls the stored activation record and avoids reprocessing. Jess™, therefore, is extremely efficient. To illustrate an example of the Rete network let us view the network formed from our working example and illustrate what each node represents. We can view the Rete network within Jess™ by invoking the (`view`) command (See Figure 80).

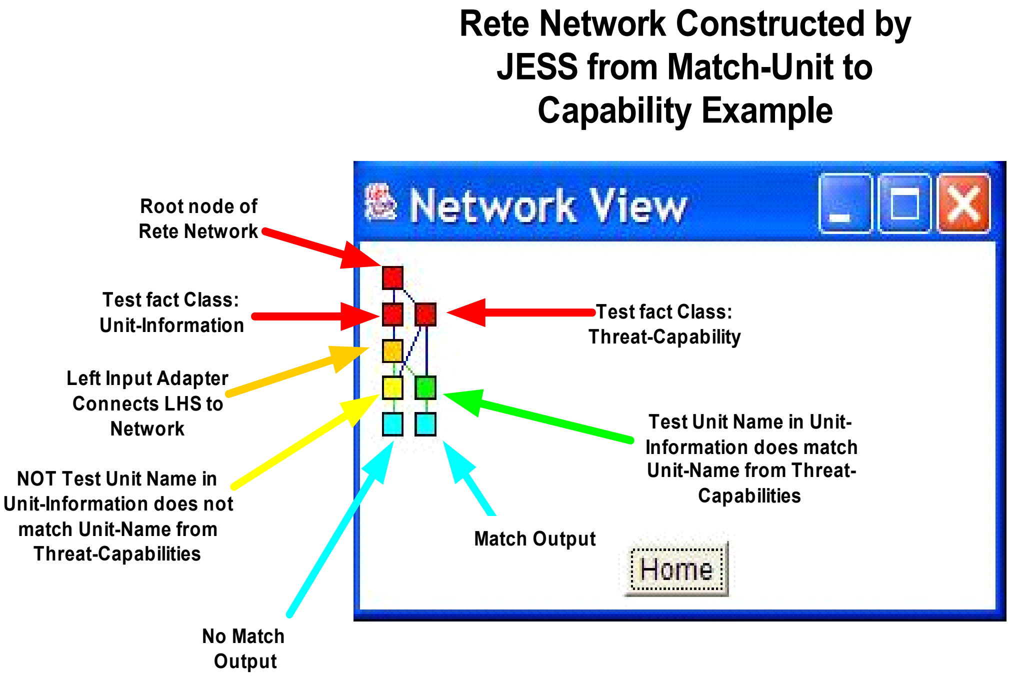

Rete Network Constructed by JESS from Match-Unit to Capability Example
Network View
Home
Root node of Rete Network
Test fact Class: Unit-Information
Test fact Class: Threat-Capability
Left Input Adapter Connects LHS to Network
NOT Test Unit Name in Unit-Information does not match Unit-Name from Threat-Capabilities
Test Unit Name in Unit-Information does match Unit-Name from Threat-Capabilities
Match Output
No Match Output

Figure 80. Rete Network Formed from Match-Unit-Capability Example.

Our example admittedly demonstrated the most basic functionality of Jess™. The real power of Jess™can only be realized when interacting with large numbers of facts and a multitude of different and complex rules. Implementing Jess™ in conjunction with a SWEB application, can, and will likely yield some interesting and powerful possibilities. Providing Jess™ has a focused and bounded usage, great efficiencies and capabilities can be gained when implemented in conjunction with a SWEB application.

D. KNOWLEDGE BASE DESIGN

The design criteria for a knowledge base are just as open ended and dependent upon the requirements of the application as any data model, application or program. There are general design considerations that continually resurface and are important enough to step through in our discussion about KBs. Much of the design criteria and considerations follow the patterns of Object Oriented Design. Before beginning our analysis of KB design criteria let us establish the fact that we are approaching the design of a KB required to serve a *specific* domain purpose. We want to differentiate this type of KB from one that may be used to serve a more general role. The domain oriented KB will be the variety likely to be encountered in military applications and will be focus for our design discussion. That said, much as the design criteria of an ontology, the content of the KB should be defined and populated with only relevant content designed to serve its purpose (Russel, 1999, 218).

1. Modular Approach

The knowledge base should be designed with modularity in mind. As we shall see there are advantages to modularity not only in the physical components such as the separation of the KB from the inference engine, but also in the knowledge stores or content of the KB. Since KBs can become very complex, the principal of modularity is essential to facilitate decomposing the KB into smaller parts - each of which may be debugged, upgraded and performance tuned separately (Booch, 1994, 16). The more complex a system is the more open it is to complete and total breakdown (Peter, 1986, 153), and we can ill afford to have the brains of our SWEB application cease to function. It then will be no different than any other web application.

2. Knowledge Base/Inference Engine

The modular principles of KB design should ideally separate the inference engine from the KB when possible (Russell, 1995, 218). This division allows the system to enjoy a separation of concerns by enabling two critical functions of the KB to be loosely coupled and independent. The alternative could be a critical dependency in which both the KB and the inference engine become helplessly inseparable, making debugging, performance and maintenance more difficult. The modular approach allows the creator of the KB to focus on building the content and developing the KR, while being less concerned about how the inference engine will interact with it (Russel, 1995, 218). Of course, the KB creator must ensure the KR, knowledge architecture and the supporting data structures are able to be manipulated by the inference procedure with maximum efficiency.

3. Knowledge Clusters

Now that we have addressed the modularity of the components of the physical KB let us shift our attention to the KB content to see the potential advantages of making the content modular[112].

Before we launch into this discussion, it must be understood there are many philosophies regarding the aggregation and dis-aggregation of KB content and supporting data structures. Each philosophy has strengths and weaknesses and is largely dependent on the application's requirements. Along with the aggregation philosophies there are also philosophies that have emerged on the storage of knowledge in large generic receptacles versus smaller, more numerous and specific knowledge stores. Again, each philosophy has particular merits and we deliberately chose not to address these issues here. For our KB discussion, as we stated above, we will focus on specific knowledge, stated as generally as possible, and its supporting data stores residing on a network designed to support the decision maker in a specific problem space. The generic and more general KBs would likely have slightly different design criteria and is beyond our scope. Later, when we discuss a practical application we will show how a generic data source can be

[112] See Wachmuth and Gangler. Knowledge Packets and Knowledge Packet Structures (1991). information

enabled and leveraged in conjunction with more specific knowledge content to augment a SWEB application. With that focus in mind, let us proceed with our discussion on knowledge modularity.

To reduce complexity within the knowledge content of the KB, the ontology and supporting data structures must be designed with modularity in mind. By organizing the knowledge content into small clusters that support some aspect of the domain, they can be developed independently and linked to other supporting knowledge or data structures as required (Russel, 1999, 218). To achieve this goal a functional decomposition of the problem space can be accomplished to breakdown the content to its atomic state and to the minimum acceptable dis-aggregation that can be tolerated and still solve a portion of the problem space. This is the idea of task-specific problem solving (Wachsmuth, 1990, 1). Wachsmuth and Gangler develop this idea and formalize it calling it Domain Oriented Knowledge Structuring (DOKS). The DOKS principals structure the knowledge content into knowledge packages or modules which are essentially small, autonomous KBs oriented to solving specific problems (Wachsmuth, 1991, 2). We will discuss DOKS and the concept of structuring knowledge by module in its own section as we feel it is an important and worthy concept.

4. Loosely Coupled

In keeping with the DOKS concepts introduced in the previous chapter let us look at how KB content functions and interacts with other content by maintaining the loosely coupled nature it was designed for.

As we stated, the KBs of the SWEB will be dependent upon the network. As such the content may be discovered, retrieved and stored on demand via the network. With this being the case, the ontologies that represent the content and their relationships within the KB must be flexible enough to function in an information/knowledge on demand environment. As we asserted earlier, the KBs of the SWEB, while not required to possess all content as their ES predecessors, they must at least store the foundational/background knowledge required to support and augment the information/knowledge gained from the network. To function in this capacity the KB must enable its foundational/background propositions to support multiple knowledge modules (Wachmuth, 1991, 3). The module concept can function in this capacity because the modules were designed to be autonomous, but linkable. In OO terms this is called loosely coupled. To illustrate this point, understand that the modules themselves may contain knowledge usable by another module that can be linked by any number of modules requiring that knowledge. Within the KB a "small networked world" is being formed as modules continue to link.

The repeated linking of modules internal and external to the KB also creates redundancy within the KB itself. Multiple links are established to relevant knowledge through adjacent terms and concepts, reuse and extensions of ontologies. The more links occurring the less prone the knowledge structures are to catastrophic failure if the network or software fails (Kurzweil, 1999, 288). A redundancy as well as a necessary functionality is being built into to the KB link by link.

The modules can assist this linking process by communicating their usability conditions through a Knowledge Module Ontology. When their usability conditions are not met the module will be partitioned or invisible to the ASKing procedure. Suffice it to say that the SWEB requires this effect to occur on a large scale for it to reach its full potential (See Figure 81).

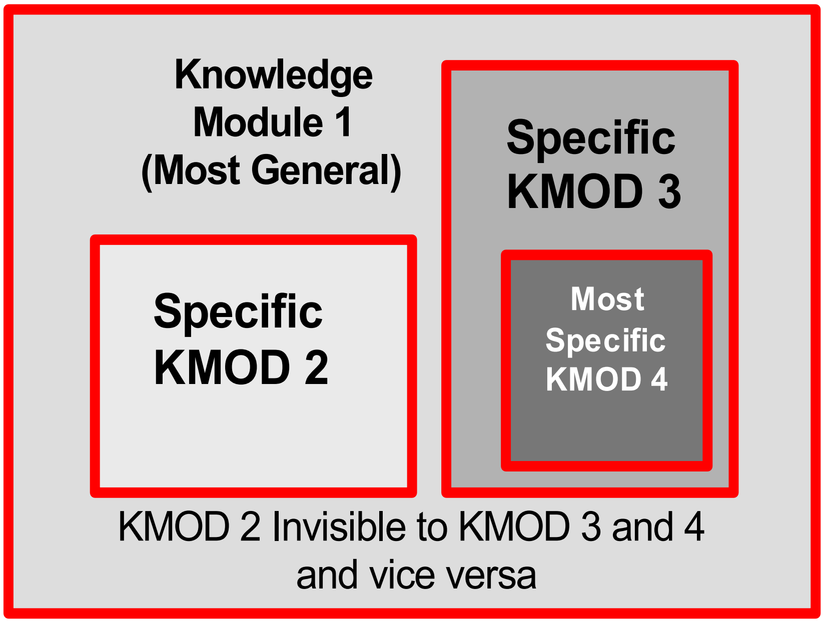


Figure 81. Example Knowledge Module Visibility (After: Wachmuth, 1991).

5. Network

We cannot emphasize enough how important the network is to the KB of a given SWEB application. Because of the network's central role the function of the KB will undoubtedly be adversely impacted if the network connection to the Wide Web fails. Planning for this failure, several network specific design choices for the KB must be considered for the KBs of the SWEB to function.

Since much of the knowledge required by SWEB applications will reside in the network, there is no need for the KB to possess and store the required source in total. In fact one of the economies gained by a networked environment is the availability of usable resources without the cost of ownership (Parsons, 2003, 5). The KB and its proxies must be designed to access these external sources routinely and provide a repository for a local cache of that data, information or knowledge for further processing. The local cache will also serve as a backup to support KB functions dependent on the networked source if the network (connection) goes down. While the cache will not be as current as the networked source[113] it will still provide the KB with a workable contingency in the event of a network failure.

These local caches within the KB are where the inference, ASKing functions and any other conversion or manipulation of the original resource that is required occurs (See Figure 82). All processing now conducted on the cached content is independent of the original networked source. Transformations of the content stored in the local cache can be executed to turn content into instances of the ontology and propositions of the KB, while manipulating only the content contained in a copy of the cache. By executing all transformations and manipulations on a copy of the cache, the original cache can be used to track the pedigree or traceability of the content from the original sources.

In the Distributed Computing and Data Sources chapters, we discussed many of the details involving how external sources are drawn into the KB and transformed as required to support the operations of the application. This is a critical function of the KB and the requirements for a reliable interface to the network must be thoroughly considered.

[113] The currency of the cache is dependent upon how often snapshots of the original source are taken.

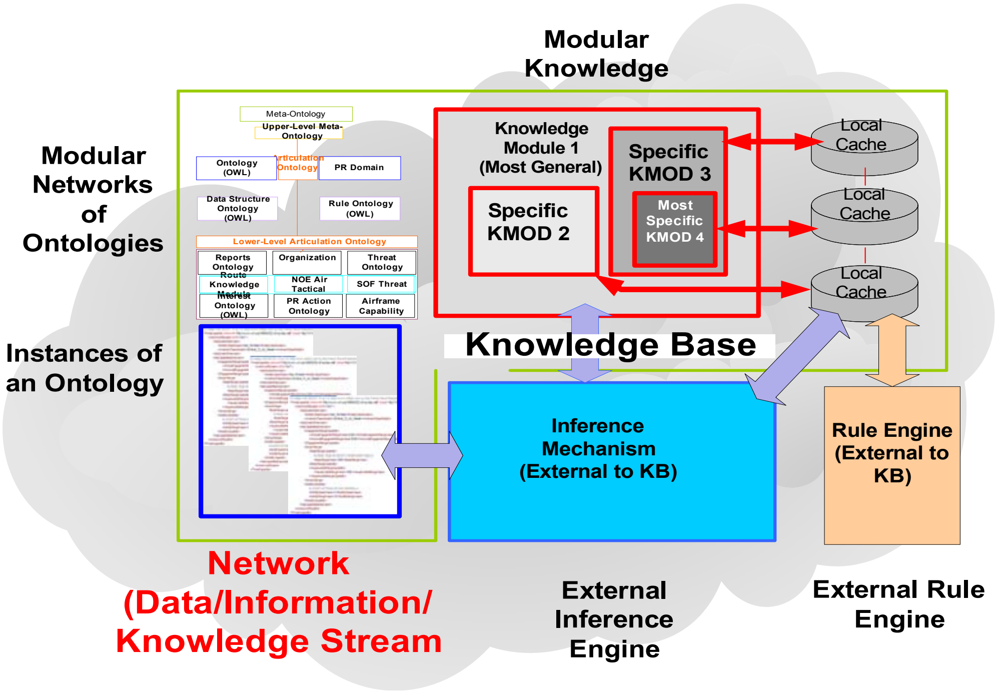


Figure 82. SWEB Network Knowledge Base Architecture.

E. KNOWLEDGE BASE ORGANIZATION

The need for large scale KBs is unavoidable for complex problem sets (Wachmuth, 1991, 5). As one might expect the more complex the problem space the potentially greater the demand for large knowledge stores and the supporting data sources to understand the problem space. As we alluded to above, the principles of Domain Oriented Knowledge Structuring (DOKS) were developed to model knowledge to solve complex problems. In fact Wachmuth and Gangler took the DOKS concepts and applied it to one of the most complex problems we have yet to solve, Natural Language Processing (NLP). They tried to develop a text understanding system supported by a large KB containing their semantic background. If the DOKS concepts can work in problem spaces such as NLP, let us see how it scales to the complex problems of the Military Domain.

1. Domain Oriented Knowledge Structuring (DOKS)

a. Knowledge Modules

We mentioned above that DOKS orients knowledge to a problem space or sub-problem space of a domain. The knowledge oriented to the domain is organized into Knowledge Packets or Knowledge Modules based on their function. Throughout this work we will refer to them as Knowledge Modules, Modules or K-Mods. A K-Mod is a component of the KB made up of Knowledge Elements. We can equate Knowledge Elements to the facts of the KB and K-Mods to the sentences or propositions that facts comprise. Collections of Knowledge Elements relevant to a given problem space belong to a K-Mod. If we dissect a K-Mod we will find a set of Knowledge Elements and their propositions based on an understanding of the problem space in which it is oriented (Wachmuth, 1991, 6). K-Mods can also contain other, more specific K-Mods. A K-Mod as we mentioned above is a small, fully functioning autonomous KB that can solve a domain specific problem. A KB can contain many K-Mods. One of the challenges to a KB with many K-Mods is selecting the right K-Mod for the right purpose.

b. Knowledge Packet Structures

Knowledge Packet structures are the Meta-Ontology(ies) that establish the usability conditions for determining what K-Mod is suited to handle what problem. The usability conditions contained within the K-Mod also assert the rules to be used by the inference procedure. The Knowledge Packet Structures serve as a point of entry to the K-Mods and may be a likely role for an agent to occupy. When a specific K-Mod is called by the KB, other K-Mods that have no applicability to the operation are made invisible by the Knowledge Packet Structure, thus restricting operations to the relevant K-Mods. This concept is called locality of reasoning. This aspect of DOKS replicates the manner in which humans use knowledge. The Knowledge Packet Structure serves as the connector and recalls the grouped knowledge found in the one or more relevant K-Mods that meets the usability conditions for a given situation and partitions the K-Mods that do not (See Figure 83). The visible K-Mods are normally small enough to be tractable to human users and lend themselves to easier maintenance by subscribing to a modular, loosely coupled design.

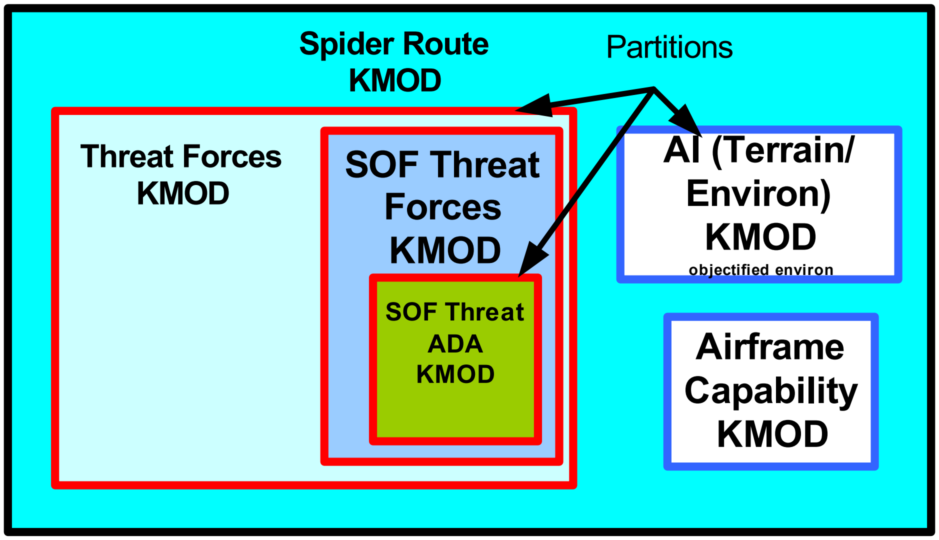


Figure 83. Spider Route Knowledge Module with Partitions.

F. SUMMARY

The Knowledge Bases (KBs) of SWEB applications, much like their predecessors in Expert Systems (ES), will serve an important function. But, unlike their ES predecessors, the KBs of the SWEB leverage the connectedness of the vast networks of the World Wide Web (WWW) to return previously unthinkable efficiencies. The WWW will obviate the requirements for a KB to possess, maintain and store its knowledge sources to replicate expert human decision making. Instead, information will be discovered, retrieved, and enabled through deep interoperability, shared meaning, distributed computing and a network of data/information/knowledge sources. With this the KBs of the SWEB will provide data, information and knowledge on demand and supply it to KB for additional processing such as reasoning. Only local caches or mirrors will be required to safeguard against network failures and non-availability of sources. The KB of the SWEB must be designed so its physical architecture and knowledge/data structures are modular and loosely coupled to mitigate complexity, foster reuse and extensibility, and expose its knowledge to other applications. The experts generally agree that an ontology plus instances is a KB, but we would argue the KB of the SWEB requires the addition of a network to the definition. The presently largely untapped knowledge richness of the networks of the WWW with the addition of powerful inferencing mechanisms when they arrive, combined with widespread OWL conceptualized domains, will combine to give us deep meaning and unprecedented functionality. This will finally give us the first real taste of the value the SWEB will contribute to assisting our military forces in gaining the knowledge superiority they require.

VIII. CONCLUSION

A. CONCLUSION

1. Refined Hypothesis

At the onset of this research effort we formulated two potential hypotheses to ground our assumptions and anchor our analysis. Both hypotheses in their original form remain valid, but as a result of our research we formulated a new, more insightful hypothesis we term Hypothesis C. The refined hypothesis follows:

a. Hypothesis C

The transition to the SWEB will be gradual, paced by the progression of the development and maturity of its building block technologies such as the Web Ontology Language (OWL), Agents, SWEB Services as well as Integrated Development Environments (Tools) to produce and validate such technologies. As the technologies mature applications will be implemented and a gradual proliferation will occur. Efficiencies and benefits of these partial implementations will be realizable and serve as the impetus to drive the technologies lagging in development and maturity. The value of the SWEB will then begin to be created. This proliferation and deployment of SWEB technologies will appear linear in nature and will continue to appear linear until adequate numbers of adopters connect and link their applications. The military and government domains will likely be early adopters and act as catalysts to the widespread adoption of the SWEB. At a point (Tipping Point) the adoption of the SWEB will diverge from a linear growth pattern to a non-linear, exponential growth rate until the growth reaches equilibrium or saturation. At equilibrium the SWEB will assume the commodity stature of the current WWW and the adoption of the SWEB will be complete. By necessity, much like the current web, industry, military and government agencies must adopt to communicate and remain interoperable.

B. RESEARCH QUESTIONS

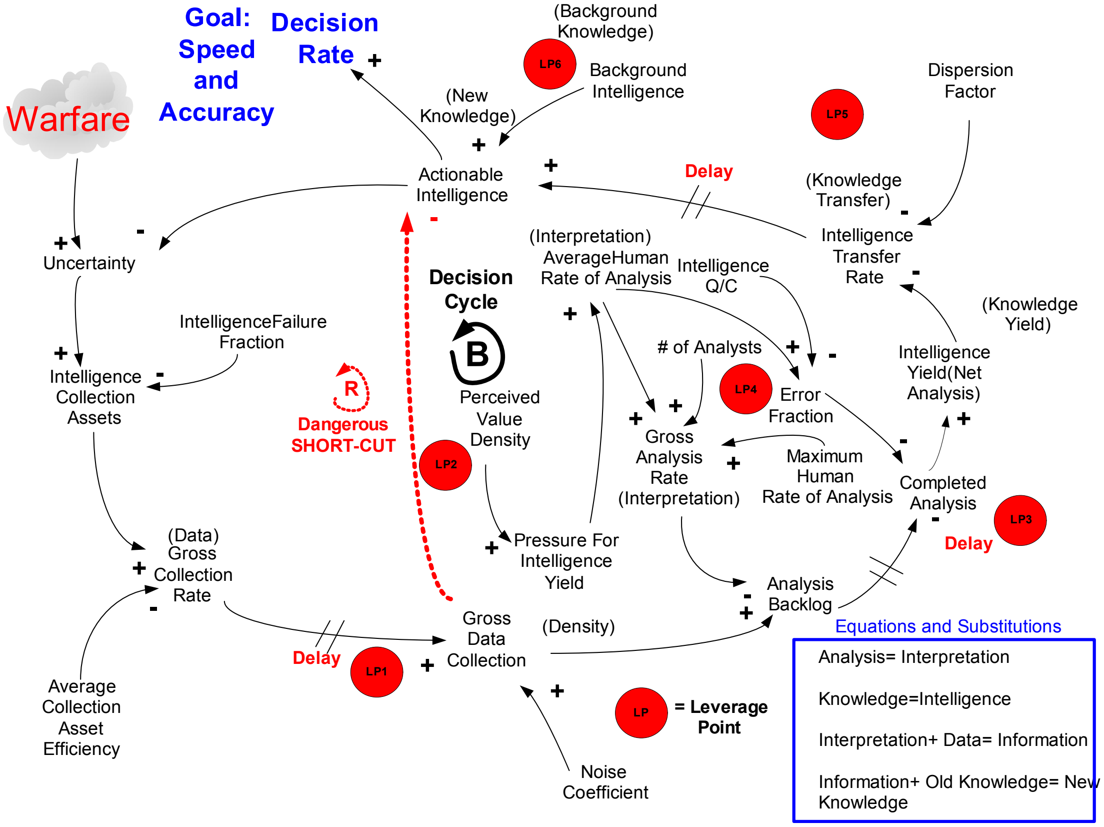
To support our hypothesis we crafted our analysis around three overarching research questions. We arrived at the overarching research questions by combining the more focused research questions answered in the individual chapters. Our overarching research questions can be found in Table 12. To reveal our conclusions to these questions we will refine our Military Decision Making Causal Loop Diagram (See Figure 84) from the Introduction and apply SWEB Technologies at the leverage points (See Figure 85).

Overarching Research Questions
How can SWEB Technologies be applied (Combat Multiplier) in the Military Domain?
What is the value added for SWEB technologies in the Military Domain?
What are the limitations/potential adoption inhibitors of SWEB technologies in the Military Domain?

Table 12. Overarching Research Questions.

C. MODEL

To review, Figure 84 depicts our model from the introduction. The potential leverage points we identified with the red circles will be the medium by which we will reveal our conclusions. We should note while we expect efficiencies to be gained by applying SWEB technologies, the technologies themselves are not expected to mitigate the inefficiencies (delays/bottlenecks) entirely. Additional efficiencies must be recouped by reengineering the processes and instituting organizational, procedural and cultural changes in the human users.



Analysis= Interpretation

Knowledge=Intelligence

Interpretation+ Data= Information

Information+ Old Knowledge= New Knowledge

Figure 84. Military Decision Making CLD and Potential Leverage Points.

1. Analysis of Refined Model with Enabled Leverage Points

a. Leverage Point 1 (LP1) Apply KR and Domain Theory

Leverage Point 1 was identified between the interaction of Data Collection Rate and Gross Data Collection. The delay depicts the Collection Delay originating from the collection of undescribed, potentially unorganized data without meaning from the collection assets. The implementation of an SWEB Application begins by enabling the data source through rigorous Knowledge Representation (KR) derived from a domain ontology. To interject a degree of intelligence and preprocessing at the sensor by embedding meaning in the data upon collection will make data self describing. If the collected data is now self describing, fewer resources are required to pre-sort and disseminate the data to the analyst(s) responsible for interpreting the data. The data can describe its own machine interpretable content. Additionally, the Noise Coefficient which we described as interjecting valueless background clutter into Gross Data

Collection can be mitigated because the noise will not possess a self describing aspect and can be discarded before it enters the system. This will reduce the system load at the point of entry, effectively shortening the delay.

b. Leverage Point 2 (LP2) Apply Machine Reasoning and Rules

Leverage Point 2 is the dynamics resulting from the Perceived Value Density of the Decision Maker and the Pressure for Intelligence Yield imposed by the system as a consequence. This pressure dynamic will always exist to a degree, but pressure can only be exerted on the human analyst. By introducing enabled data from the point of entry, machines can now assist humans in the interpretation or analysis process. The Pressure for Intelligence Yield can now be shared by machines and humans, but we expect the pressure to be less severe since the machines will work at a higher rate of analysis (classifying, comparing, and correlating) as compared to a human analyst. The human will now have some of the menial analysis completed by a machine and be able to focus on improving the analytical quality by leveraging human cognitive skills, not yet reproducible by machines. While unable to remove all temptations to take the dangerous shortcut and admit raw data from the Gross Data Collection into Actionable Intelligence, it will substantially reduce the temptation. This will mitigate the negative effect on Actionable Intelligence and not slow the Net Decision Rate resulting from the pollution of Actionable Intelligence by raw unevaluated data.

c. Leverage Point 3 (LP3) Apply Agents and Machines

LP3 is targeted at the Analysis Delay largely caused by quantity of data versus the Maximum Human Rate of Analysis (MHRA). The MHRA is a result of the human cognitive limits. The introduction of KR-enabled data, able to be interpreted by machines and agents, will now offload some of the menial analysis tasks from the human and allow the human to focus on more advanced human reasoning increasing the quality of the analysis. The personnel who once filled the roles the machines now occupy can be reallocated to quality control and afforded the ability to apply deeper, cognitive analysis to the materials.

d. Leverage Point 4 (LP4) Apply Computer Reasoning and Rules

LP 4 is targeted at a potential unanticipated side effect from MHRA and the effects of Pressure for Intelligence Yield. The more rapidly a human analyzes material the greater the fatigue factor. This however this is not true of machines. The agent and machines assigned to assist humans with analysis have a higher degree of endurance and in fact cannot be fatigued. Therefore, if fatigue does not occur, the Error Fraction caused by fatigue and workload is largely removed from the system. Humans can now be employed in quality control roles vice employed in the repetitive, mundane roles better suited to a machine. The most important point from LP4 is the fact that the human can analyze at a certain rate, for a certain amount of time, but machines and software agents are not susceptible to this limitation.

e. Leverage Point 5 (LP5) Apply Agents and Services

LP5 focuses on the transfer delay in the Intelligence Transfer Rate between Intelligence Yield and Actionable Intelligence. The delay is caused by latency in the network, network traffic (bottlenecks/collisions) and network availability. By transferring self describing content through agents and SWEB Services will allow more precise distribution and delivery by incorporating a publication and subscription system. This will effectively reduce the intrinsic Dispersion Factor of knowledge, in this case our surrogate Intelligence, by transferring content that is self describing as part of a service or agent system. Rigorous KR will mitigate the dispersion factor before it can affect Intelligence Transfer Rate to its intended recipient as the intended recipients now have agents capable watching for changes in content their decision maker is interested in at the point of entry, countering the dispersion factor.

f. Leverage Point 6 (LP6) Apply KB and Knowledge Construction

LP 6 is found in the added value of relevant, usable and available Background Intelligence (Background Knowledge) to create new Actionable Intelligence (New Knowledge). By integrating data driven inference and reasoning, background knowledge can be brought to a new level of quality making the New Knowledge Equation: Old Knowledge + Information = New Knowledge execute more efficiently by providing higher quality, machine interpretable background knowledge on demand.

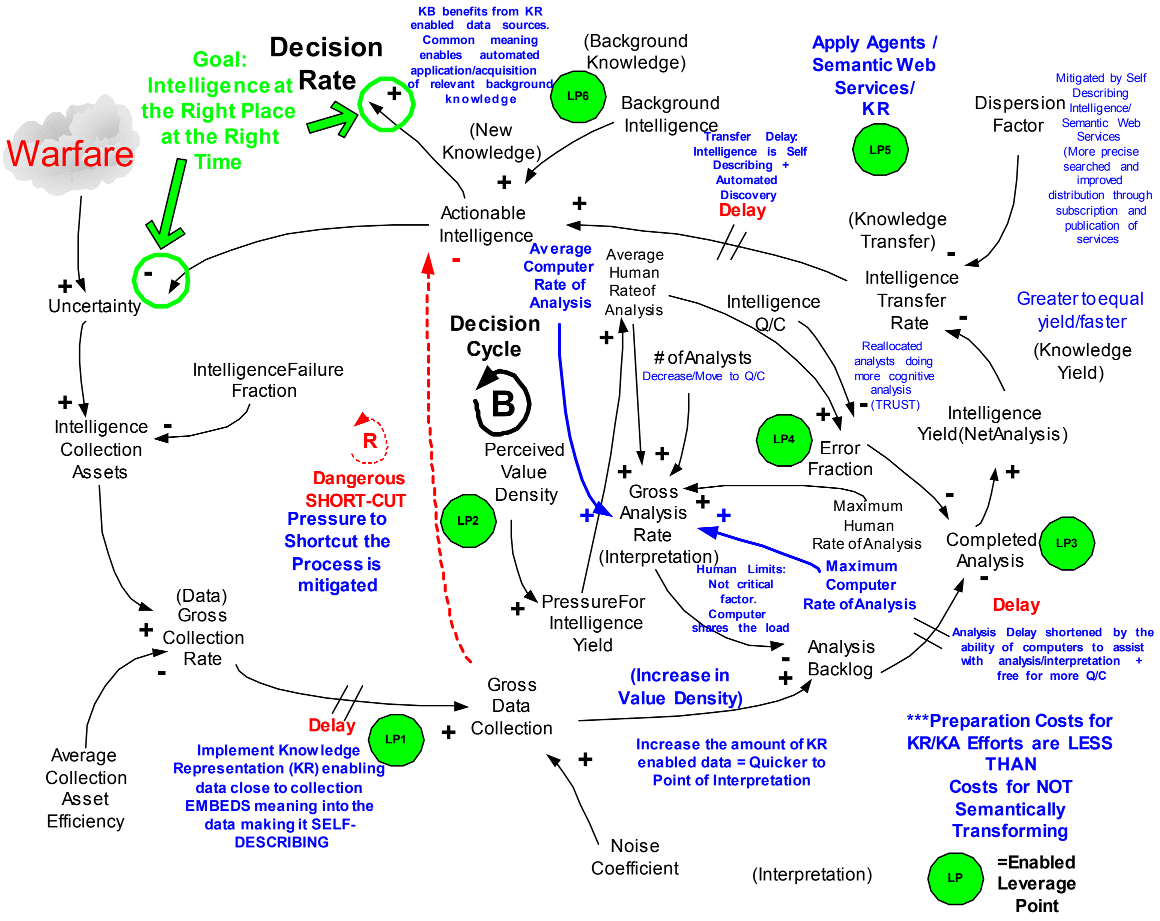

Warfare
Goal:
Intelligence at the Right Place at the Right Time
Decision Rate
KB benefits from KR enabled data sources. Common meaning enables automated application/acquisition of relevant background knowledge
(Background Knowledge)
LP6
Background Intelligence
(New Knowledge)
Actionable Intelligence
Apply Agents / Semantic Web Services/ KR
LP5
Transfer Delay: Intelligence is Self Describing + Automated Discovery
Delay
Dispersion Factor
Mitigated by Self Describing Intelligence/ Semantic Web Services (More precise searched and improved distribution through subscription and publication of services
(Knowledge Transfer)
Intelligence Transfer Rate
Greater to equal yield/faster
(Knowledge Yield)
Uncertainty
Average Computer Rate of Analysis
Average Human Rateof Analysis
Intelligence Q/C
Decision Cycle
B
# ofAnalysts
Decrease/Move to Q/C
Reallocated analysts doing more cognitive analysis
(TRUST)
IntelligenceFailure Fraction
Intelligence Collection Assets
R
Dangerous SHORT-CUT
Pressure to Shortcut the Process is mitigated
Perceived Value Density
LP2
LP4
Error Fraction
Intelligence Yield(NetAnalysis)
Gross Analysis Rate (Interpretation)
Maximum Human Rate ofAnalysis
Maximum Computer Rate ofAnalysis
Human Limits: Not critical factor. Computer shares the load
Completed Analysis
LP3
Delay
Analysis Delay shortened by the ability of computers to assist with analysis/interpretation + free for more Q/C
(Data) Gross Collection Rate
PressureFor Intelligence Yield
Analysis Backlog
(Increase in Value Density)
Average Collection Asset Efficiency
Delay
LP1
Implement Knowledge Representation (KR) enabling data close to collection EMBEDS meaning into the data making it SELF-DESCRIBING
Gross Data Collection
Increase the amount of KR enabled data = Quicker to Point of Interpretation
***Preparation Costs for KR/KA Efforts are LESS THAN Costs for NOT Semantically Transforming
Noise Coefficient
(Interpretation)
LP
=Enabled Leverage Point
+
-

Figure 85. Military Decision Making CLD with Enabled Leverage Points.

2. Overall Model

As demonstrated from a review of the refined model many of the potential leverage points identified in the model we presented in the Introduction are suited to application of SWEB Technologies to gain speed, accuracy and precision. While we know the bottlenecks and delays can be mitigated to a level, we further understand without changes in the decision making process, people and leadership the efficiencies cannot be gained. The adoption of the SWEB will be realized and soon machines will prove to be of greater value to war fighting. When machines are able to interpret the content they process and further assist humans by classifying, correlating and comparing content before the human analysis begins, their potential will be further realized. This off-loading, or delegation, will produce faster sensor-to-shooter times and assist in achieving the speed required to achieve victory on any battlefield.

D. FUTURE WORK

1. Integration of All the Components

While all components can operate separately, the real power of the SWEB will not be realized until all are operating seamlessly in concert. The integration of all the components of the SWEB is the first demonstration of the value the SWEB is capable of providing. Our research efforts ended with independently functioning pieces; an integrated application is left for future research efforts. See Figure 86 for a proposed Generic Semantic Web Application Architecture Pattern illustrating likely integration requirements.

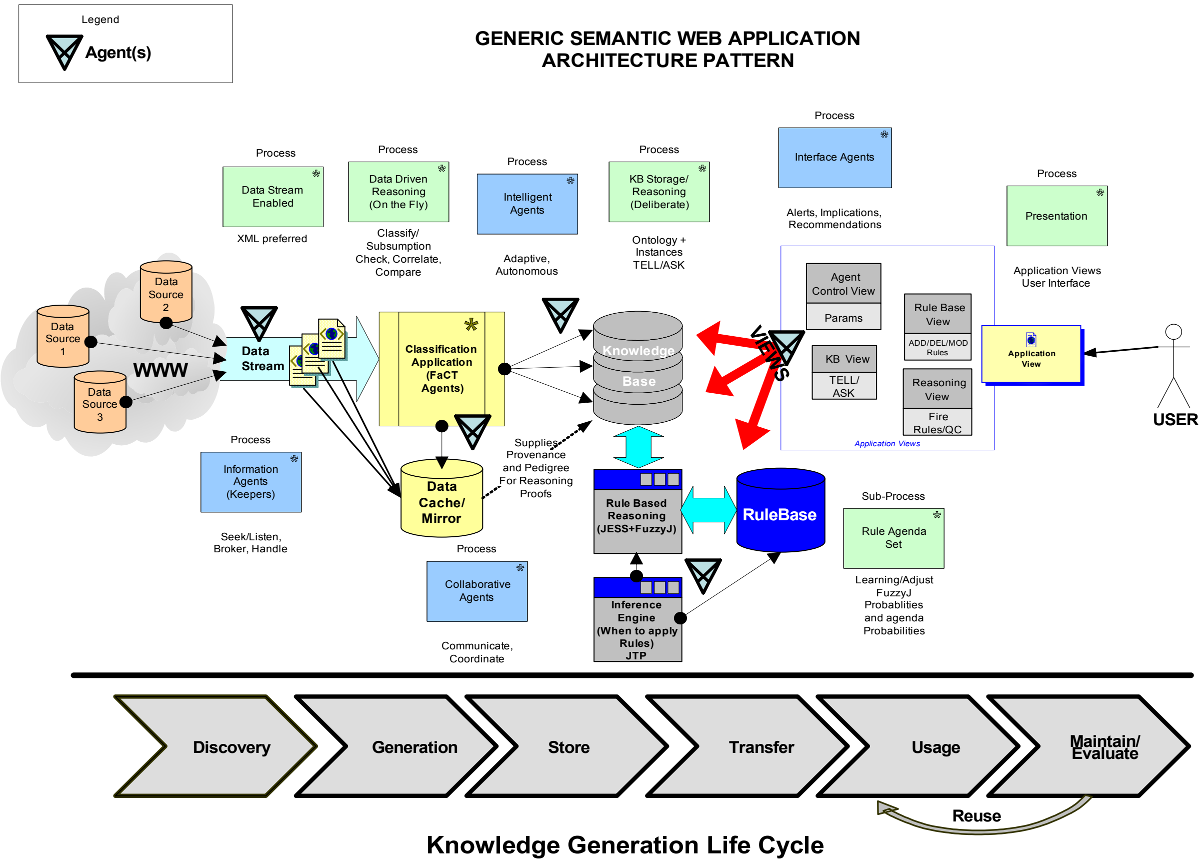

Legend
Agent(s)
GENERIC SEMANTIC WEB APPLICATION ARCHITECTURE PATTERN
Process
Data Stream Enabled
XML preferred
Process
Data Driven Reasoning (On the Fly)
Classify/ Subsumption Check, Correlate, Compare
Process
Intelligent Agents
Adaptive, Autonomous
Process
KB Storage/ Reasoning (Deliberate)
Ontology + Instances TELL/ASK
Process
Interface Agents
Alerts, Implications, Recommendations
Process
Presentation
Application Views User Interface
Data Source 2
Data Source 1
Data Source 3
WWW
Data Stream
Classification Application (FaCT Agents)
Knowledge Base
VIEWS
Agent Control View
Params
KB View
TELL/ ASK
Rule Base View
ADD/DEL/MOD Rules
Reasoning View
Fire Rules/QC
Application Views
Application View
USER
Process
Information Agents (Keepers)
Seek/Listen, Broker, Handle
Data Cache/ Mirror
Supplies Provenance and Pedigree For Reasoning Proofs
Process
Collaborative Agents
Communicate, Coordinate
Rule Based Reasoning (JESS+FuzzyJ)
RuleBase
Inference Engine (When to apply Rules) JTP
Sub-Process
Rule Agenda Set
Learning/Adjust FuzzyJ Probablities and agenda Probabilities
Discovery
Generation
Store
Transfer
Usage
Maintain/ Evaluate
Reuse
Knowledge Generation Life Cycle

Figure 86. Integrated Semantic Web Architecture (Generic).

2. Trust/Security

Unquestionably, trust and security are critical to the implementation of the SWEB within the military domain. Due to the complex nature and detail required to properly address these issues they were intentionally left for future research.

3. Transition Guidance

The various functioning pieces are generically grouped under the ArchAngel project discussed in the Agents chapter. Code for ArchAngel is slated to be deployed on a server and its supporting infrastructure at NPS. The server will be controlled and administered by Mr. Doug Horner of the Expeditionary Pervasive Sensing (EPS) research group. The current thread will build upon a Personnel Recovery (PR) scenario with emphasis on dynamic route planning. The design and development of the ontologies and the external rule based system is underway by a contractor under the guidance of Mr. Horner. Anyone interested in participating in this effort is encouraged to contact Mr. Horner.

APPENDIX. GLOSSARY

Term	Description
.NET	- Set of Microsoft technologies designed for developing interconnected applications. - Similar to J2EE's purpose
AA	- Antiaircraft
AI	- Artificial Intelligence
API	- Application Programming Interface
Backward Chaining	- Given a goal state, a system that first checks to see if the goal matches the initial facts given, if not the system looks for rules whose conclusion matches the goal. - The system looks to support the goal with known facts/concepts from the KB
Binary Predicate	- Questions that have two arguments - Such as properties
BML	- Battlefield Management Language
$C\phi$	- Common Knowledge
Classes	- Abstraction mechanism for grouping resources with similar characteristics - Every OWL class is associated with a set of individuals or its class extension
CLD	- Causal Loop Diagram
Closed World Assumption	- A fact is false unless it has been explicitly stated as true
CoABS	- Control of Agent Based Systems
Conceptualization	- The formal structure of reality as perceived, organized and described - Includes the vocabulary and actual occurrences of a specific situation
COP	- Common Operational Picture
$D\phi$	- Distributed Knowledge
DARPA	- Defense Advanced Research Projects Agency
DDL	- Data Definition Language
DL	- Description Logic
DoD	- Department of Defense
DOKS	- Domain Oriented Knowledge Structuring
DOM	- Document Object Model. - An interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. - Document can be further processed and the results of that processing can be incorporated back into the presented page. - Reads entire document into memory.

Term	**Description**
	- Provides random access
Domain (x)	- The set of objects that may serve as inputs to a function - All possible values that can be inserted into a column is the domain on an attribute
Domain/ Universe of Discourse	- Set of entities we want to express/represent knowledge about - Bounded area on interest
ER Diagram	- Entity Relationship Diagram
ES	- Expert System
Extrinsic Properties	- Not forming and essential part of an entity - Such as the name of a weapon system or the unit it belongs to
FA	- Feasibility Assessment
Facets	- Slot constraints - Value types - Allowable values
FaCT Reasoner	- Fast Classifications of Terminologies
Forward Chaining	- System begins with a fact assertion and attempts to establish conclusions based on the new facts - Data driven/Data Directed - Forward chaining slowly builds up a general picture of the problem space as new facts are asserted
Frame	- The information relevant to a particular concept stored in a single complex entity - OWL class axiom
GH5	- Generic Hub 5 - NATO Data model to facilitate data sharing
IFF	- Information Flow Framework
Instances	- Concepts denoting single items rather than sets or categories - Individuals in a class extension are called instances
Intrinsic Properties	- Situated or belonging solely to; forms an essential part of an entity - Such as the engagement range of a weapon system
IT	- Information Technology
J2EE	- The Java 2 Platform, Enterprise Edition (J2EE) - Sun Microsystems' standard for developing multi-tier enterprise applications. - J2EE is a Java-based technology stack, built on top of J2SE (Java 2 Platform, Standard Edition) - Primarily targeted to server-side applications
Jess	- Java Expert System Shell
KB	- Knowledge Base
KR	- Knowledge Representation
KVA	- Knowledge Value Added
Lexical (Lexicon)	- Of or related to words or the vocabulary of a language as distinguished from its grammar and construction

Term	**Description**
LHS	- Left Hand Side
LP	- Leverage Point
MHRA	- Maximum Human Rate of Analysis
Monotonic	- New information cannot retract/counter previous information
Ontology	- Explicit, formal specifications of the concepts in the domain and the relationships among them
OOB	- Order of Battle
OODA Loop	- The natural human decision making cycle - Observe, Orient, Decide, Act - Invented by COL (Ret) Boyd
OOP	- Object Oriented Programming
OWL	- Web Ontology Language - Designed for use by applications that need to process the content of information instead of just presenting information to humans - Facilitates greater machine readability of Web content than that supported by XML, RDF, and RDF Schema by providing additional vocabulary along with a formal semantics. - Has three increasingly-expressive sub-languages: OWL Lite, OWL DL, and OWL Full.
PSI	- Published Subject Indicator - XTM community concept also applicable to OWL meaning an authoritative source
Range (y)	- The set of objects that may serve as the value of a function - All possible values that may be inserted into a tuple or row - Allowed classes for slots
RDF	- Resource Description Framework - Language for representing information about resources in the World Wide Web. - Particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. - OWL is built on top of the RDF recommendation
Reification	- To regard something abstract as a material concrete thing
RHS	- Right Hand Side
SAX	- Simple API for XML - Similar to DOM, but “looks through” each node of the XML document based on event stream. - Does not load entire document into memory - Does not provide random access
SD	- Systems Dynamics
Semantic Network	- A system that represents objects as nodes of a graph with taxonomic structure

Term	Description
	- Subclass and instance relations may be used to derive new information not explicitly stated (subsumption check/classification) - Allows for efficient inheritance
Slots	- Important properties - Relationships, attributes, procedures
SOAP	- Simple Object Access Protocol
Subsumption	- Classifying an entity/concept under a more general category - A result of this classification is that the entity inherits the attributes of the more general category
SVG	- Scaleable Vector Graphics
SWEB	- Semantic Web
Unary Predicates	- Questions that have one argument - Such as a class
URI	- Uniform Resource Identifier
W3C	- World Wide Web Consortium - Develops interoperable technologies (specifications, guidelines, software, and tools). - A forum for information, commerce, communication, and collective understanding. - Open standards organization whose scope is the World Wide Web.
WSDL	- Web Services Definition language
WWW	- World Wide Web
XHTML™	- Extensible Hyper-Text Markup Language - A family of current and future document types and modules that reproduce, subset, and extend HTML, reformulated in XML. - Family document types are all XML-based, and ultimately are designed to work in conjunction with XML-based user agents. - Successor of HTML - A series of specifications has been developed for XHTML. - It is well-formed XML that uses the HTML tag set.
XML:DB	- Initiative to create and implement standards for interfacing with and interacting with XML databases - Functionally, comparable to JDBC and ODBC for relational database connectivity
XPath	- A W3C Recommendation that describes a syntax for selecting a set of nodes from an XML document. (Hunter, 2001, 619) - Provides basic facilities for manipulating strings, numbers and booleans. - Uses compact, non-XML syntax for use of XPath within URIs and XML attribute values. - Operates on the abstract, logical structure of an XML document, rather than its surface syntax.

Term	**Description**
	- Gets its name from its use of a path notation as in URLs for navigating through hierarchical structure of an XML document. - In addition to use for addressing, is also designed so that it has a natural subset that can be used for matching (testing whether or not a node matches a pattern); this use of XPath is described in XSLT.
XQuery	- XML Query Language - Designed to provide features for retrieving and interpreting information from XML sources, using concise and easily understood queries. - Flexible enough to query a broad spectrum of XML information sources, including both databases and documents - Derived from an XML query language called Quilt, which in turn borrowed features from several other languages, including XPath 1.0 and SQL among others.
XSL	- Extensible Stylesheet Language - Language for expressing stylesheets - It consists of three parts: XSL Transformations (XSLT): a language for transforming XML documents, the XML Path Language (XPath), and XSL Formatting Objects: an XML vocabulary for specifying formatting semantics - An XSL stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Adams, Thomas K. *FutureWarfare and the Decline of Human Decision-Making* (2001). Parameters, Winter 2001-02, pp. 57-71.

Alberts, D. S., Garstka, J. J., Stein, F. P. (1999). Network Centric Warfare: Developing and Leveraging Information Superiority. DoD C4ISR Cooperative Research Program.

Ahmed, K., Ancha, S., Cioroianu, A., Cousins, J., Crosbie, J., Davies, J., Gabhart, K., Gould, S., Laddad, R., Li, S., Macmillan, B., Rivers-Moore, D., Skubal, J., Watson, K., Williams, S., Hart, J. (2001). Professional Java XML. Birmingham, U.K: Wrox Press Ltd.

Ankolenkar, A., Burstein, M., Paolucci, M., Payne, T., Sycara, K., Lassila, O., McIlraith, S., Cao Son, T., Zeng, H., Hobbs, J., Martin, D., Narayanan, S., and McDermott, D. DAML-S: Semantic Markup for Web Services. Report Dated 5 May 2003.

Arquilla, J., Ronfeldt, D. (1997). In Athena's Camp. RAND.

Ayala, D., Browne, C., Chopra, V., Sarang, P., Apshankar, K., McAllister, T. (2002). Professional Open Source Web Services. Birmingham, U.K.: Wrox Press Ltd.

Bachimont, B., Isaac, A., Troncy, R. Semantic Commitment for Designing Ontologies: A Proposal. (2002). In Knowledge Engineering and Knowledge Management: Ontologies for the Semantic Web, 13th Annual Conference, Siguenza, Spain, October 2002.

Bechhofer, Sean (2001). OilEd 3.4 Manual. Department of Computer Science: University of Manchester, U.K.

[2] Bechhofer, S., Horrocks, I., Goble, C., Stevens, R. (2001). OilEd: A Reasonable Ontology Editor for the Semantic Web. Information Management Group, Department of Computer Science: University of Manchester, U.K.

Beckett, D., Grant, Jan. (W3C SWAD Report, 23 January 2003). Semantic Web Scalability and Storage: Mapping Semantic Web Data with RDBMSes. [Online]. Available: [http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/], March 17, 2003.

Berners-Lee, T., Hendler, J., Lassila,O. (2001). The Semantic Web. Scientific American [Online]. Available: [http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2], July 9, 2003.

[2] Berners-Lee, T., Connolly, D., Swick, R. (1999). Web Architecture: Describing and Exchanging Data. [Online]. Available: [www.w3.org/1999/04/WebData], July 9, 2003.

Bertino, E., Catania, B., Zarri, G. P. (2001). Intelligent Database Systems. San Francisco, California: Addison-Wesley.

Bequet, H., Kunnumpurath, M., Rhody, S., Tost, A. (2002). Beginning Java Web Services Birmingham, UK: Wrox Press Ltd.

Birbeck, M., Duckett, J., Gudmundsson, O., Kobak, P., Lenz, E., Livingstone, S., Marcus, D., Mohr, S., Pinnock, J., Visco, K., Watt, A., Williams, K., Zaev, Z., Ozu, N. (2002). Professional XML, 2nd Edition Birmingham, UK: Wrox Press Ltd.

Booch, G. Object-Oriented Analysis and Design with Applications. (2001). San Francisco, California: Addison-Wesley.

Borenstein, J., Fox, J. (2003). Web Services Journal. SYS-CON Publishing. Semantic Discovery for Web Services: A Step Toward Fulfillment of the Vision.

Cardon, A., Durand, S. A Model of Crisis Management System including Mental Representations. (1997) *In Proceedings of AAAI Spring Symposium*.

Campbell, K. E., Oliver, D. E., Spackman, K. A., Shortliffe, E. H. Representing Thoughts, Words and Things in the UMLS. (1998).

Carey, S. A. Battle Management Language (BML): A Simulation to C4I (SIMCI Perspective (White Paper, 4 November 2002). Northrop Grumman Information Technology.

[2] Carey, S. A., Kleiner, M. S., Hieb, M. R. Brown, R. Development of a C2 Standards of Task Representation for C4ISR Systems, Simulations and Robotics: Battle Management Language.

Daconta, M., Obrst, L., Smith, K. (2003). The Semantic Web: A Guide to the Future of XML, Web Services and Knowledge Management. Indianapolis, Indiana: Wiley Publishing Inc.

Davenport, T. H., Prusak, L. (1998). Working Knowledge: How Organizations Manage What They Know. Boston, Massachusetts: Harvard Business School Press.

Davies, J., Fensel, D., Van Harmelen, F. (2003). Towards the Semantic Web: Ontology Driven Knowledge Management. West Sussex, U.K.: John Wiley and Sons.

Dean, M. (2002) Guest Lecture SW 4599 Automated Hardware/Software Integration in DoD. Naval Postgraduate School, 20 November 2002 (Video Teleconference).

Deitel, P. J. (2002). Java, How to Program, Fourth Edition. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Description Logics. (2003) Whitepaper from Network Inference Holdings Ltd.

Description Logic Handbook. (2003). Eds Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. Cambridge, U.K.: Cambridge Press.

Edwards, W. K. (1999). Core Jini. Prentice-Hall, Inc.: Upper Saddle River, New Jersey.

[2] Edwards, W. K. Tom Rodden. (2001). Jini, Example by Example. Upper Saddle River, New Jersey: Prentice-Hall, Inc

Fagin, R., Halpern, J. Y., Moses, Y., Vardi, M. Y. (1995) Reasoning About Knowledge. Cambridge, Massachusetts: MIT Press.

Fernandez, M., Perez, A., Vicente, A. Towards a Method to Conceptualize Domain Ontologies. (unk). Universidad Politécnica de Madrid.

Fikes, R., McGuinness, D. (unk). Creating, Maintaining, and Integrating Understandable Knowledge Bases. KSL Labratories, Stanford University.

Freidman-Hill, E. (2003). Jess in Action: Rule Based Systems in Java. Greenwich, Connecticut: Manning Publications Company.

[2] Friedman-Hill, E. (2003). Jess, The Expert System Shell for the Java Platform (Version 6.1RC1). Distributed Computing Systems, Sandia National Laboratories: Livermore: California.

Fowler, M. Analysis Patterns: Reusable Object Models. (1997). Menlo Park, California:Addison Wesley.

Gardenfors, P. (2000). Conceptual Spaces: The Geometry of Thought. Cambridge, Massachusets: MIT Press.

Gell-Mann, M. (1997). The Simple and the Complex. In Alberts, D. S., Czerwinski, T. J. (Eds.), Complexity, Global Politics, and National Security. National Defense University.

Gladwell, M. (2002). The Tipping Point: How Little Things Can Make a Big Difference. New York: Back Bay Books.

Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., Nakamura, Y., Neyama, R. (2002). Building Web Services with Java™. Sams Publishing, Indianapolis, Indiana.

Gruber, T. R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. (Revision 23 August 1993). In Formal Ontology in Conceptual Analysis and Knowledge Representation. (Eds.) Guarino, N., Poli, R. [Online]. Available: [http://ksl.stanfors.edu/KSL 93-04], March 17, 2003.

Hayek, F. A. The Use of Knowledge in Society. Reprinted from the *American Economic Review, XXXV,* No. 4; September 1945, pp. 519-30.

[1] Hayek, F.A. Economics and Knowledge. *Reprinted from* Economica IV (new ser., 1937), pp. 33-54.

Horner, D. Semantic Web (SWEB) Integration in ESG Enabling Experimentation (EEE). 2002. Naval Postgraduate School, Monterey, California.

Horrocks, Ian (unk). The FaCT System. Medical Informatics Group, Department of Computer Science: University of Manchester, U.K.

Hunter, D., Cagle, K., Dix, C., Kovack, R., Pinnock, J., Rafter, J. (2002). Beginning XML, 2nd Edition. Birmingham, UK: Wrox Press Ltd.

Jasper, R., Uschold, M. (1999?). A Framework for Understanding and Classifying Ontology Applications. Boeing Math and Computing Technology: Seattle, Washington.

(Jini AO) Jini™ Architectural Overview: Technical White Paper. Sun Microsystems. © 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303 U.S.A.

(Jini EO) Jini™ Network Technology: An Executive Overview. Sun Microsystems. © 2001 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303 U.S.A.

(Jini Now?) Why Jini™ Technology Now? Sun Microsystems. © 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303 U.S.A.

(Jini TE) Jini™ Technology and Emerging Network Technologies. Sun Microsystems. © 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303 U.S.A.

Jini vs. Web Services - Ein Leistungsvergleich. Tobias Schwaegli. September 2002. ETH Zurich, 8092 Zurich, Switzerland.

Johnson, S. (2001). Emergence: The Connected Lives of Ants, Brains, Cities and Software. New York: Scribner.

Joint Vision 2020. (2000). Office of Primary Responsibility: Director for Strategic Plans and Policy, J5, Strategy Division.

Kahn, M., Cicalese, C., Brake, D., Glahe, A., Brill, D., Tsurutani, B., Ito, J., Combs, V. (2002) DARPA CoABS Grid Users Manual. Version 4 – Draft, October 2002. Global InfoTek, Inc. 1920 Association Drive, Suite 200, Reston, Virginia 20191.

Kim, H. M. Exploiting Small-Worlds of the Sematic Web using Mata-Data Based Shared Ontologies: Using Structure and Semantics to Connect Heterogeneous, Local Ontologies. (2002). In Proceedings of WITS.

Klein, M. Combining and Relating Ontologies: An Analysis of Problems and Solutions. Amsterdam, The Netherlands: Vrije Universiteit, Division of Mathematics and Computer Science.

[2] Klein, M. Supporting Evolving Ontologies on the Internet. Amsterdam, The Netherlands: Vrije Universiteit, Division of Mathematics and Computer Science.

Knott, Anne M. Knowledge Dynamics: Reconciling Competing Hypotheses from Economics and Sociology. University of Pennsylvania: The Wharton School of Business. (2001).

Kumaran, S., Kumaran, I. Jini Technology: An Overview. (2001). Prentice Hall, Inc. Upper Saddle River: New Jersey.

Kurzweil, R. The Age of Intelligent Machines (1990). MIT:MIT Press.

Levesque, H., Lakemeyer, G. (2000). The Logic of Knowledgebases. Massachusetts: MIT Press.

Maedche, A. (2002). Ontology Learning for the Semantic Web. Boston, Massachusetts: Kluwer Academic Publishers.

[2] Maedche, A., Staab, S., Stojanovic, N., Studer, R., Sure, Y. Semantic Portal: The SEAL Approach. (2003). In Spinning the Semantic Web: Brining the World Wide Web to its Full Potential. Eds Fensel, D., Hendler, J., Lieberman, H., Wahlster, W. Massachusetts: MIT Press.

Marakas, G. M. (1999). Decision Support Systems in the 21st Century. New Jersey: Prentice Hall.

Marinescu, D.C. (2002). Internet-Based Workflow Management: Toward a Semantic Web. New York: John Wiley and Sons.

McGuinness, D. L., Van Harmelen, F. (W3C Working Draft, 31 March 2003). Web Ontology Language (OWL): Overview [Online]. Available: [http://www.w3.org/TR/owl-features/], March 17, 2003.

[2] McGuinness, D. Conceptual Modeling for Distributed Ontology Environments. (2000). In Proceedings of Eighth International Conference on Conceptual Structures Logic, Linguistic, and Computational Issues 14-18 August 2000.

Mitra, P., Wiederhold, G., Kersten, M. (2000). A Graph-Oriented Model for Articulation of Ontology Interdependencies. Stanford University, CWI.

Moczar, L., Aston, J. Cocoon Developers Handbook. (2002). Sams Publishing, Developer's Library, 201 West 103rd Street, Indianapolis, Indiana 46290.

Motter, A. E., deMoura, A., Lai, Y. C., Dasgupta, P. (2002) *Topology of the Conceptual Network of Language.* The American Physical Society, Vol. 65, 065102- (1-4).

Nagappan, R., Skoczylas, R., Sriganesh, R. Developing Java Web Services. (2003). Indianapolis, Indiana: Wiley Publishing, Inc.

Nardi, D., Brachman, R. (2003). An Introduction to Description Logics. In Description Logic Handbook. Eds. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. Cambridge, U.K.: Cambridge Press.

Network Inference. (2003). Description Logics (White Paper).

Nissen, M. E. (2002). An Extended Model of Knowledge-Flow Dynamics. Published in Communications of the Association for Information Systems (Volume 8) 251-266.

[2] Nissen, M., Kamel, M., Sengupta, K. (2000). Integrated Analysis and Design of Knowledge Systems and Processes. In Information Resources Management Journal, January-March 2000.

Noy, N. F, McGuinness, D. L. Ontology Development 101: A Guide to Creating Your First Ontology. [Online]. Available: [http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf], March 17, 2003.

O'Dell, C., Grayson, C. J. Jr. (1998). If Only We Knew What We Know. New York: The Free Press.

Ogden, C. K., Richards, I. A. (1923). The Meaning of Meaning. New York: Harcourt, Brace and World Inc.

Omelayenko, Borys. (No Date). RDFT: A Mapping Meta-Ontology for Business Integration. Amsterdam, The Netherlands: Vrije Universiteit, Division of Mathematics and Computer Science.

Obrst, L., Liu, H. (2003). Knowledge Representation, Ontological Engineering , and Topic Maps. In Park, J., Hunting, S. (Eds.), XML Topic Maps: Creating and Using Topic Maps for the Web.

Parsons, David (2003). Introductory Java. London, U.K.: Continuum.

Patel-Schneider, P., Hayes, P., Horrocks, I. (W3C Working Draft, 31 March 2003). OWL Web Ontology Language Semantics and Abstract Syntax [Online]. Available: [http://www.w3.org/TR/owl-features/], March 17, 2003.

Peter, L. (1986). The Peter Pyramid. New York: William Morrow.

Peterson, B., Stine, J., Darken, R. (submitted). *Eliciting Knowledge from Military Ground Navigators*. In Berndt Brehmer, Rannan Lipshitz, and Henry Montgomery (Eds.), How Professionals Make Decisions. Washington D.C.: American Psychological Association.

Pohl, J. Adapting to the Information Age. (2000). First Published at InterSymp 2000, Bin Baden, Germany.

[2] Pohl, J. Information Centric Decision-Support Systems: A Blueprint for 'Interoperability'. (2000).

Professional XML 2nd Edition. (2001). Wrox Press Ltd:U.K.

Riedl, Reinhard. (2002). Some Critical Remarks in Favour of IT-Based Knowledge Management. In Knowledge Management and Information Technology, Vol. III, No.1, February 2002.

Roberts, Angus. An Introduction to OilEd: OilEd Version 3.3. Department of Computer Science: University of Manchester, U.K.

Russel, S., Norvig, P. (1995). Artificial Intelligence: A Modern Approach. New Jersey: Prentice Hall.

Schmitt, John F. Command and (Out of) Control: The Military Implications and the Complexity Theory. In Complexity, Global Politics and National Security (Eds.) Alberts, D. S., Czerwinski, T. J. Institute for National Strategic Studies: National Defense University, 1997.

Shaw, M. L. G., Gaines, B. R. (1995) Comparing Conceptual Structures: Consensus, Conflict, Correspondence and Contrast. University of Calgary: Knowledge Science Institute.

Shaw, M. L. G., Gaines, B. R. Kelly's "Geometry of Psychological Space" and its Significance for Cognitive Modeling. The New Psychologist, 23-31 October, 1992.

Sivashanmugam, K., Verma, K., Sheth, A., Miller, J. (unk). Adding Semantics to Web Services Standards. Large Scale Distributed Information Systems (LSDIS) Lab, Department of Computer Science, University of Georgia.

Smith, M. K., Welty, C., McGuinness, D. (W3C Working Draft, 10 February 2003). Web Ontology Language (OWL) Guide (Version 1.0) [Online]. Available: [http://www.w3.org/TR/2003/WD-owl-guide-20030210/], March 17, 2003.

Sowa, J. F. (2000). Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pacific Grove: Brooks/Cole.

Stine, J. (2000) Representing Tactical Land Navigation Expertise. Master's Thesis, Naval Postgraduate School, Monterey, California.

Sterman, J. D. (2003) Business Dynamics: Systems Thinking and Modeling for a Complex World. McGraw-Hill, United States.

Sure, Y., Studer, R. (2003). A Methodology for Ontology-Based Knowledge Management. In Towards the Semantic Web: Ontology Driven Knowledge Management. Eds. Davies, J., Fensel, D., Van Harmelen. West Sussex, U.K.: John Wiley and Sons.

Tanenbaum, A., van Steen, M. (2002). Distributed Systems: Principles and Paradigms. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Vanderschraaf, Peter. "Common Knowledge", *The Stanford Encyclopedia of Philosophy (Summer 2002 Edition)*, Edward N. Zalta (ed.), URL = [http://plato.stanford.edu/archives/sum2002/entries/common-knowledge], May 13, 2003.

Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., Stein, L. A. (W3C Working Draft, 21 February 2003). Web Ontology Language (OWL) Reference (Version 1.0) [Online]. Available: [http://www.w3.org/TR/owl-ref/], March 17, 2003.

Vatant, B. Topic Maps from Representation to Identity: Conversation, Names, and Published Subject Indicators. (2003) In XML Topic Maps (Eds.), Park, J., Hunting, S. SanFrancisco: Addison-Wesley.

Von Clausewitz, C. On War. (1976). Princeton, New Jersey: Princeton University Press.

W3C Semantic Web. [Online]. Available: [http://www.w3.org/2001/sw/#pub], July 9, 2003.

Watts, D. (1999). Network, Dynamics, and the Small World Phenomenon. In American Journal of Sociology, Volume 105, Issue 2, (September 1999), pp. 493-597.

Wachsmuth, I., Gangler, B. (1991) Knowledge Packets and Knowledge Packet Structures. In O.Herzog & C.R. Rollinger (Eds.), Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence.

Weiss, G. (2001). Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence. Cambridge: Massachusetts Institute of Technology.

Williams, K., Barnes, D., Likes, B., Mohr, S., Morris, P., Novick, A., Polshaw, A., Sabin, S., Tennison, J. (2002). SQL Server XML Distilled. U.K.: Curlingstone Publishing, Ltd.

XML.com [http://www.xml.com/pub/a/2001/06/20/databases.html], July 8, 2003, Author: Igor Dayen.

XML.com [http://www.xml.com/pub/a/2002/01/09/xmldb_api.html], July 8, 2003, Author: Kimbro Staken.

XML.com Introduction to Native XML Databases, [http://www.xml.com/pub/a/2001/10/31/nativexmldb.html], July 8, 2003, Author: Kimbro Staken.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dan C. Boger
   Naval Postgraduate School
   Monterey, California

4. Alex Bordetsky
   Naval Postgraduate School
   Monterey, California

5. Kim Swecker
   SPAWAR
   San Diego, California

6. Douglas P. Horner
   Naval Postgraduate School
   Monterey, California

7. Sam Chance
   Naval Postgraduate School
   Monterey, California

8. Marty Hagenston
   Naval Postgraduate School
   Monterey, California